Subject: Accelerating a one-line program doing matrix multiplication
Posted by on Mon, 27 Sep 2010 09:18:43 GMT
View Forum Message <> Reply to Message

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!

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Wed, 29 Sep 2010 15:24:23 GMT
View Forum Message <> Reply to Message

On Sep 29, 4:49 pm, chris <rog...@googlemail.com> wrote:
> 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

Thanks!!

Great, I did not know about this construction, and honestly I do not
understand how it works (is there any documentation about it?).
Anyways, I tried it, and unfortunately I saw that it needed ~20%
longer (the complete function, not the rebin only). So, it is not
faster.. but it is great though.

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by pgrigis on Wed, 29 Sep 2010 15:34:29 GMT
View Forum Message <> Reply to Message

>
> [skip]

>
> 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

Another option is:
a#(fltarr(5)+1)

But I would expect this to be slower...

Ciao,
Paolo

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by penteado on Wed, 29 Sep 2010 15:45:56 GMT
View Forum Message <> Reply to Message

On Sep 29, 12:24 pm, Axel M <axe...@gmail.com> wrote:
> Great, I did not know about this construction, and honestly I do not
> understand how it works (is there any documentation about it?).
> Anyways, I tried it, and unfortunately I saw that it needed ~20%
> longer (the complete function, not the rebin only). So, it is not
> faster.. but it is great though.

It is replicating a structure of a single field which contains the
array input ({temp:input}), then selecting only a single field (the

first, 0) of the resulting structure array. Documentation for this
would be on creation and use of structures.

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Wed, 29 Sep 2010 15:55:33 GMT
View Forum Message <> Reply to Message

On 29 Sep., 17:45, Paulo Penteado <pp.pente...@gmail.com> wrote:
> On Sep 29, 12:24 pm, Axel M <axe...@gmail.com> wrote:
>
>> Great, I did not know about this construction, and honestly I do not
>> understand how it works (is there any documentation about it?).
>> Anyways, I tried it, and unfortunately I saw that it needed ~20%
>> longer (the complete function, not the rebin only). So, it is not
>> faster.. but it is great though.
>
> It is replicating a structure of a single field which contains the
> array input ({temp:input}), then selecting only a single field (the
> first, 0) of the resulting structure array. Documentation for this
> would be on creation and use of structures.

Ok, I got it. Thanks! Then probably it is the memory allocation for
the array of structures which takes so long... it would be great if
the ITT people would develop a _fast_ vector replicate, I fear
rebinning is not the best option.

In any case, based on the answers, I assume that my problem is rather
on the matrix multiplication part, so I can probably do nothing for
that.

Thanks a lot

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by pgrigis on Wed, 29 Sep 2010 16:05:58 GMT
View Forum Message <> Reply to Message

On Sep 29, 11:55 am, Axel M <axe...@gmail.com> wrote:
> On 29 Sep., 17:45, Paulo Penteado <pp.pente...@gmail.com> wrote:
>
>> On Sep 29, 12:24 pm, Axel M <axe...@gmail.com> wrote:
>
>>> Great, I did not know about this construction, and honestly I do not
>>> understand how it works (is there any documentation about it?).
>>> Anyways, I tried it, and unfortunately I saw that it needed ~20%
>>> longer (the complete function, not the rebin only). So, it is not

>>> faster.. but it is great though.
>
>> It is replicating a structure of a single field which contains the
>> array input ({temp:input}), then selecting only a single field (the
>> first, 0) of the resulting structure array. Documentation for this
>> would be on creation and use of structures.
>
> Ok, I got it. Thanks! Then probably it is the memory allocation for
> the array of structures which takes so long... it would be great if
> the ITT people would develop a _fast_ vector replicate, I fear
> rebinning is not the best option.
>
> In any case, based on the answers, I assume that my problem is rather
> on the matrix multiplication part, so I can probably do nothing for
> that.
>
> Thanks a lot

well considering your original problem - you need to apply
a linear transformation to N vectors v_i=(x_i,y_i,z_i),
for i going from 0 to a large N, right?

I would just explicitely compute the transformed vectors

z_i=(xx_i,yy_i,zz_i)

by just writing out in the program the computation for every
component,
i.e.

xx=x*c1+y*c2+z*c3+c4
and same for yy,zz with appropriate constant coefficients c1,c2,c3,c4
(that are the same for all i).

But then maybe i misunderstood the problem...

Ciao,
Paolo

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by Karl[1] on Wed, 29 Sep 2010 16:57:05 GMT
View Forum Message <> Reply to Message

On Sep 29, 10:05 am, Paolo <pgri...@gmail.com> wrote:
> On Sep 29, 11:55 am, Axel M <axe...@gmail.com> wrote:
>

>
>
>> On 29 Sep., 17:45, Paulo Penteado <pp.pente...@gmail.com> wrote:
>
>>> On Sep 29, 12:24 pm, Axel M <axe...@gmail.com> wrote:
>
>>>> Great, I did not know about this construction, and honestly I do not
>>>> understand how it works (is there any documentation about it?).
>>>> Anyways, I tried it, and unfortunately I saw that it needed ~20%
>>>> longer (the complete function, not the rebin only). So, it is not
>>>> faster.. but it is great though.
>
>>> It is replicating a structure of a single field which contains the
>>> array input ({temp:input}), then selecting only a single field (the
>>> first, 0) of the resulting structure array. Documentation for this
>>> would be on creation and use of structures.
>
>> Ok, I got it. Thanks! Then probably it is the memory allocation for
>> the array of structures which takes so long... it would be great if
>> the ITT people would develop a _fast_ vector replicate, I fear
>> rebinning is not the best option.
>
>> In any case, based on the answers, I assume that my problem is rather
>> on the matrix multiplication part, so I can probably do nothing for
>> that.
>
>> Thanks a lot
>
> well considering your original problem - you need to apply
> a linear transformation to N vectors v_i=(x_i,y_i,z_i),
> for i going from 0 to a large N, right?
>
> I would just explicitely compute the transformed vectors
>
> z_i=(xx_i,yy_i,zz_i)
>
> by just writing out in the program the computation for every
> component,
> i.e.
>
> xx=x*c1+y*c2+z*c3+c4
> and same for yy,zz with appropriate constant coefficients c1,c2,c3,c4
> (that are the same for all i).
>
> But then maybe i misunderstood the problem...
>
> Ciao,
> Paolo

Yeah, I think you are right.

Another way to see it:

```
FUNCTION vc2rc, v0,v1,v2,v3,vc
      xform = [[v1],[v2],[v3]]
      n = <number of points in vc>
      for i=0, n-1
         temp = vc[*,i]
         temp = temp # xform + v0
         vc[*,i] = temp
      end
END
```

This assumes that you can change vc itself and that v0 is a 3-vector. In this case, there is only one copy of the point array, as it is being transformed in place. In other schemes, there may have been as many as three or four copies. If it is not OK to change vc, then this function would have to make a vr array of the same shape as vc and return it. But it is still the best solution as far as memory goes.

Yeah, the for loop is going to be slow, but a test will tell if it is faster than other approaches. If the program causes paging to disk with the original approach, then the for loop may be faster. If speed is really, really important, then the above can be implemented in a C DLM.

And yes, the three lines with "temp" can be collapsed into one, but IDL will make small temps anyway here and so a single line may not be much faster. I left it as three lines for clarity.

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Thu, 30 Sep 2010 08:39:59 GMT
View Forum Message <> Reply to Message

On Sep 29, 6:57 pm, Karl <karl.w.schu...@gmail.com> wrote:
> On Sep 29, 10:05 am, Paolo <pgri...@gmail.com> wrote:
>
>
>
>> On Sep 29, 11:55 am, Axel M <axe...@gmail.com> wrote:
>
>>> On 29 Sep., 17:45, Paulo Penteado <pp.pente...@gmail.com> wrote:
>
>>>> On Sep 29, 12:24 pm, Axel M <axe...@gmail.com> wrote:
>

>>>> > Great, I did not know about this construction, and honestly I do not
>>>> > understand how it works (is there any documentation about it?).
>>>> > Anyways, I tried it, and unfortunately I saw that it needed ~20%
>>>> > longer (the complete function, not the rebin only). So, it is not
>>>> > faster.. but it is great though.
>
>>>> It is replicating a structure of a single field which contains the
>>>> array input ({temp:input}), then selecting only a single field (the
>>>> first, 0) of the resulting structure array. Documentation for this
>>>> would be on creation and use of structures.
>
>>> Ok, I got it. Thanks! Then probably it is the memory allocation for
>>> the array of structures which takes so long... it would be great if
>>> the ITT people would develop a _fast_ vector replicate, I fear
>>> rebinning is not the best option.
>
>>> In any case, based on the answers, I assume that my problem is rather
>>> on the matrix multiplication part, so I can probably do nothing for
>>> that.
>
>>> Thanks a lot
>
>> well considering your original problem - you need to apply
>> a linear transformation to N vectors v_i=(x_i,y_i,z_i),
>> for i going from 0 to a large N, right?
>
>> I would just explicitely compute the transformed vectors
>
>> z_i=(xx_i,yy_i,zz_i)
>
>> by just writing out in the program the computation for every
>> component,
>> i.e.
>
>> xx=x*c1+y*c2+z*c3+c4
>> and same for yy,zz with appropriate constant coefficients c1,c2,c3,c4
>> (that are the same for all i).
>
>> But then maybe i misunderstood the problem...
>
>> Ciao,
>> Paolo
>
> Yeah, I think you are right.
>
> Another way to see it:
>
> FUNCTION vc2rc, v0,v1,v2,v3,vc

```
>         xform = [[v1],[v2],[v3]]
>         n = <number of points in vc>
>         for i=0, n-1
>            temp = vc[*,i]
>            temp = temp # xform + v0
>            vc[*,i] = temp
>         end
>  END
>
>  This assumes that you can change vc itself and that v0 is a 3-vector.
>  In this case, there is only one copy of the point array, as it is
>  being transformed in place.  In other schemes, there may have been as
>  many as three or four copies.  If it is not OK to change vc, then this
>  function would have to make a vr array of the same shape as vc and
>  return it.  But it is still the best solution as far as memory goes.
>
>  Yeah, the for loop is going to be slow, but a test will tell if it is
>  faster than other approaches.  If the program causes paging to disk
>  with the original approach, then the for loop may be faster.  If speed
>  is really, really important, then the above can be implemented in a C
>  DLM.
>
>  And yes, the three lines with "temp" can be collapsed into one, but
>  IDL will make small temps anyway here and so a single line may not be
>  much faster.  I left it as three lines for clarity.
```

Hi,

Thanks for the idea. I tried it, below is the function code (original and "accelerated" with your idea) and the test code. By explicitly applying the linear transformation (_accel version) within a loop it took 15 times longer... I guess IDL does this better with the # operator.

I still think I can most definitely gain time by using the fact that vc represents just all indexes of an array, but I have to find out how to exploit this property...

```
FUNCTION vc2rc, v0,v1,v2,v3,vc
 RETURN, [[v1],[v2],[v3]] # vc + REBIN(v0, SIZE(vc, /DIMENSIONS))
END

FUNCTION vc2rc_accel, v0,v1,v2,v3,vc
 npoints = (SIZE(vc, /DIMENSIONS))[1]
 for i=0L, npoints-1 DO BEGIN
  vc[*,i] = vc[0,i] * v1 + vc[1,i] * v2 + vc[2,i] * v3 + v0
 endfor
 RETURN, vc
```

```
END

PRO testspeed
 dims = [100,100,100]
       i = LINDGEN(LONG(dims[0])*dims[1]*dims[2]) ;image dimensions
       vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
(dims[1]))], [(i / (dims[0] * dims[1]))]])
 v0=[5,5,5] ;origin
 v1=[1.0,0,0] ;vectors
 v2=[0,1.0,0]
 v3=[0,0,2.0]

 t0 =SYSTIME(/SECONDS)
 rc = vc2rc_accel(v0,v1,v2,v3,vc)
 rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
(dims[1]))], [(i / (dims[0] * dims[1]))]])
 rc = vc2rc_accel(v0,v1,v2,v3,vc)
 print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)

 rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
(dims[1]))], [(i / (dims[0] * dims[1]))]])

 t0 =SYSTIME(/SECONDS)
 rc = vc2rc(v0,v1,v2,v3,vc)
 rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
(dims[1]))], [(i / (dims[0] * dims[1]))]])
 rc = vc2rc(v0,v1,v2,v3,vc)
 print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)

 rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
(dims[1]))], [(i / (dims[0] * dims[1]))]])

 t0 =SYSTIME(/SECONDS)
 rc = vc2rc_accel(v0,v1,v2,v3,vc)
 rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
(dims[1]))], [(i / (dims[0] * dims[1]))]])
 rc = vc2rc_accel(v0,v1,v2,v3,vc)
 print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)
END
```

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Thu, 30 Sep 2010 09:41:19 GMT
View Forum Message <> Reply to Message

On Sep 30, 10:39 am, Axel M <axe...@gmail.com> wrote:
> On Sep 29, 6:57 pm, Karl <karl.w.schu...@gmail.com> wrote:
>

>
>
>> On Sep 29, 10:05 am, Paolo <pgri...@gmail.com> wrote:
>
>>> On Sep 29, 11:55 am, Axel M <axe...@gmail.com> wrote:
>
>>>> On 29 Sep., 17:45, Paulo Penteado <pp.pente...@gmail.com> wrote:
>
>>>> > On Sep 29, 12:24 pm, Axel M <axe...@gmail.com> wrote:
>
>>>> > > Great, I did not know about this construction, and honestly I do not
>>>> > > understand how it works (is there any documentation about it?).
>>>> > > Anyways, I tried it, and unfortunately I saw that it needed ~20%
>>>> > > longer (the complete function, not the rebin only). So, it is not
>>>> > > faster.. but it is great though.
>
>>>> > It is replicating a structure of a single field which contains the
>>>> > array input ({temp:input}), then selecting only a single field (the
>>>> > first, 0) of the resulting structure array. Documentation for this
>>>> > would be on creation and use of structures.
>
>>>> Ok, I got it. Thanks! Then probably it is the memory allocation for
>>>> the array of structures which takes so long... it would be great if
>>>> the ITT people would develop a _fast_ vector replicate, I fear
>>>> rebinning is not the best option.
>
>>>> In any case, based on the answers, I assume that my problem is rather
>>>> on the matrix multiplication part, so I can probably do nothing for
>>>> that.
>
>>>> Thanks a lot
>
>>> well considering your original problem - you need to apply
>>> a linear transformation to N vectors v_i=(x_i,y_i,z_i),
>>> for i going from 0 to a large N, right?
>
>>> I would just explicitely compute the transformed vectors
>
>>> z_i=(xx_i,yy_i,zz_i)
>
>>> by just writing out in the program the computation for every
>>> component,
>>> i.e.
>
>>> xx=x*c1+y*c2+z*c3+c4
>>> and same for yy,zz with appropriate constant coefficients c1,c2,c3,c4
>>> (that are the same for all i).
>

>>> But then maybe i misunderstood the problem...
>
>>> Ciao,
>>> Paolo
>
>> Yeah, I think you are right.
>
>> Another way to see it:
>
>> FUNCTION vc2rc, v0,v1,v2,v3,vc
>> xform = [[v1],[v2],[v3]]
>> n = <number of points in vc>
>> for i=0, n-1
>> temp = vc[*,i]
>> temp = temp # xform + v0
>> vc[*,i] = temp
>> end
>> END
>
>> This assumes that you can change vc itself and that v0 is a 3-vector.
>> In this case, there is only one copy of the point array, as it is
>> being transformed in place. In other schemes, there may have been as
>> many as three or four copies. If it is not OK to change vc, then this
>> function would have to make a vr array of the same shape as vc and
>> return it. But it is still the best solution as far as memory goes.
>
>> Yeah, the for loop is going to be slow, but a test will tell if it is
>> faster than other approaches. If the program causes paging to disk
>> with the original approach, then the for loop may be faster. If speed
>> is really, really important, then the above can be implemented in a C
>> DLM.
>
>> And yes, the three lines with "temp" can be collapsed into one, but
>> IDL will make small temps anyway here and so a single line may not be
>> much faster. I left it as three lines for clarity.
>
> Hi,
>
> Thanks for the idea. I tried it, below is the function code (original
> and "accelerated" with your idea) and the test code. By explicitly
> applying the linear transformation (_accel version) within a loop it
> took 15 times longer... I guess IDL does this better with the #
> operator.
>
> I still think I can most definitely gain time by using the fact that
> vc represents just all indexes of an array, but I have to find out how
> to exploit this property...
>

```
>  FUNCTION vc2rc, v0,v1,v2,v3,vc
>        RETURN, [[v1],[v2],[v3]] # vc + REBIN(v0, SIZE(vc, /DIMENSIONS))
>  END
>
>  FUNCTION vc2rc_accel, v0,v1,v2,v3,vc
>        npoints = (SIZE(vc, /DIMENSIONS))[1]
>        for i=0L, npoints-1 DO BEGIN
>              vc[*,i] = vc[0,i] * v1 + vc[1,i] * v2 + vc[2,i] * v3 + v0
>        endfor
>        RETURN, vc
>  END
>
>  PRO testspeed
>        dims = [100,100,100]
>        i = LINDGEN(LONG(dims[0])*dims[1]*dims[2]) ;image dimensions
>        vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>        v0=[5,5,5] ;origin
>        v1=[1.0,0,0] ;vectors
>        v2=[0,1.0,0]
>        v3=[0,0,2.0]
>
>        t0 =SYSTIME(/SECONDS)
>        rc = vc2rc_accel(v0,v1,v2,v3,vc)
>        rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>        rc = vc2rc_accel(v0,v1,v2,v3,vc)
>        print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)
>
>        rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>
>        t0 =SYSTIME(/SECONDS)
>        rc = vc2rc(v0,v1,v2,v3,vc)
>        rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>        rc = vc2rc(v0,v1,v2,v3,vc)
>        print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)
>
>        rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>
>        t0 =SYSTIME(/SECONDS)
>        rc = vc2rc_accel(v0,v1,v2,v3,vc)
>        rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>        rc = vc2rc_accel(v0,v1,v2,v3,vc)
>        print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)
```

> END

Hi again,

I found a solution which is ~20% faster, receiving the dimensions of
the image directly instead of the "vc" points (since, as I said, it is
in this case where speed really becomes an issue). It is doing
additions rather than multiplications, which appears to work faster.

```
FUNCTION rc_fromimage, v0,v1,v2,v3,dims
 RETURN, REBIN(v1 # INDGEN(dims[0]), [3, dims]) + REBIN(REFORM(v2 #
INDGEN(dims[1]), 3, 1, dims[1], 1), [3, dims]) + REBIN(REFORM(v3 #
INDGEN(dims[2]), 3, 1, 1, dims[2]), [3, dims])
END
```

within testspeed, the tests looks like "rc =
rc_fromimage(v0,v1,v2,v3,dims)"

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Thu, 30 Sep 2010 11:05:36 GMT
View Forum Message <> Reply to Message

On Sep 30, 11:41 am, Axel M <axe...@gmail.com> wrote:
> On Sep 30, 10:39 am, Axel M <axe...@gmail.com> wrote:
>
>
>
>> On Sep 29, 6:57 pm, Karl <karl.w.schu...@gmail.com> wrote:
>
>>> On Sep 29, 10:05 am, Paolo <pgri...@gmail.com> wrote:
>
>>>> On Sep 29, 11:55 am, Axel M <axe...@gmail.com> wrote:
>
>>>> > On 29 Sep., 17:45, Paulo Penteado <pp.pente...@gmail.com> wrote:
>
>>>> > > On Sep 29, 12:24 pm, Axel M <axe...@gmail.com> wrote:
>
>>>> > > > Great, I did not know about this construction, and honestly I do not
>>>> > > > understand how it works (is there any documentation about it?).
>>>> > > > Anyways, I tried it, and unfortunately I saw that it needed ~20%
>>>> > > > longer (the complete function, not the rebin only). So, it is not
>>>> > > > faster.. but it is great though.
>
>>>> > > It is replicating a structure of a single field which contains the
>>>> > > array input ({temp:input}), then selecting only a single field (the
>>>> > > first, 0) of the resulting structure array. Documentation for this
>>>> > > would be on creation and use of structures.

>
>>>> > Ok, I got it. Thanks! Then probably it is the memory allocation for
>>>> > the array of structures which takes so long... it would be great if
>>>> > the ITT people would develop a _fast_ vector replicate, I fear
>>>> > rebinning is not the best option.
>
>>>> > In any case, based on the answers, I assume that my problem is rather
>>>> > on the matrix multiplication part, so I can probably do nothing for
>>>> > that.
>
>>>> > Thanks a lot
>
>>>> well considering your original problem - you need to apply
>>>> a linear transformation to N vectors v_i=(x_i,y_i,z_i),
>>>> for i going from 0 to a large N, right?
>
>>>> I would just explicitely compute the transformed vectors
>
>>>> z_i=(xx_i,yy_i,zz_i)
>
>>>> by just writing out in the program the computation for every
>>>> component,
>>>> i.e.
>
>>>> xx=x*c1+y*c2+z*c3+c4
>>>> and same for yy,zz with appropriate constant coefficients c1,c2,c3,c4
>>>> (that are the same for all i).
>
>>>> But then maybe i misunderstood the problem...
>
>>>> Ciao,
>>>> Paolo
>
>>> Yeah, I think you are right.
>
>>> Another way to see it:
>
>>> FUNCTION vc2rc, v0,v1,v2,v3,vc
>>>   xform = [[v1],[v2],[v3]]
>>>   n = <number of points in vc>
>>>   for i=0, n-1
>>>   temp = vc[*,i]
>>>   temp = temp # xform + v0
>>>   vc[*,i] = temp
>>>   end
>>> END
>
>>> This assumes that you can change vc itself and that v0 is a 3-vector.

>>> In this case, there is only one copy of the point array, as it is
>>> being transformed in place. In other schemes, there may have been as
>>> many as three or four copies. If it is not OK to change vc, then this
>>> function would have to make a vr array of the same shape as vc and
>>> return it. But it is still the best solution as far as memory goes.
>
>>> Yeah, the for loop is going to be slow, but a test will tell if it is
>>> faster than other approaches. If the program causes paging to disk
>>> with the original approach, then the for loop may be faster. If speed
>>> is really, really important, then the above can be implemented in a C
>>> DLM.
>
>>> And yes, the three lines with "temp" can be collapsed into one, but
>>> IDL will make small temps anyway here and so a single line may not be
>>> much faster. I left it as three lines for clarity.
>
>> Hi,
>
>> Thanks for the idea. I tried it, below is the function code (original
>> and "accelerated" with your idea) and the test code. By explicitly
>> applying the linear transformation (_accel version) within a loop it
>> took 15 times longer... I guess IDL does this better with the #
>> operator.
>
>> I still think I can most definitely gain time by using the fact that
>> vc represents just all indexes of an array, but I have to find out how
>> to exploit this property...
>
>> FUNCTION vc2rc, v0,v1,v2,v3,vc
>> RETURN, [[v1],[v2],[v3]] # vc + REBIN(v0, SIZE(vc, /DIMENSIONS))
>> END
>
>> FUNCTION vc2rc_accel, v0,v1,v2,v3,vc
>> npoints = (SIZE(vc, /DIMENSIONS))[1]
>> for i=0L, npoints-1 DO BEGIN
>> vc[*,i] = vc[0,i] * v1 + vc[1,i] * v2 + vc[2,i] * v3 + v0
>> endfor
>> RETURN, vc
>> END
>
>> PRO testspeed
>> dims = [100,100,100]
>> i = LINDGEN(LONG(dims[0])*dims[1]*dims[2]) ;image dimensions
>> vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>> (dims[1]))], [(i / (dims[0] * dims[1]))]])
>> v0=[5,5,5] ;origin
>> v1=[1.0,0,0] ;vectors
>> v2=[0,1.0,0]

```
>>         v3=[0,0,2.0]
>
>>         t0 =SYSTIME(/SECONDS)
>>         rc = vc2rc_accel(v0,v1,v2,v3,vc)
>>         rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>>         rc = vc2rc_accel(v0,v1,v2,v3,vc)
>>         print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)
>
>>         rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>
>>         t0 =SYSTIME(/SECONDS)
>>         rc = vc2rc(v0,v1,v2,v3,vc)
>>         rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>>         rc = vc2rc(v0,v1,v2,v3,vc)
>>         print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)
>
>>         rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>
>>         t0 =SYSTIME(/SECONDS)
>>         rc = vc2rc_accel(v0,v1,v2,v3,vc)
>>         rc = 0 & vc = TRANSPOSE([[(i MOD dims[0])], [((i / dims[0]) MOD
>>  (dims[1]))], [(i / (dims[0] * dims[1]))]])
>>         rc = vc2rc_accel(v0,v1,v2,v3,vc)
>>         print, 'Time: ', STRING(SYSTIME(/SECONDS) - t0)
>>  END
>
>  Hi again,
>
>  I found a solution which is ~20% faster, receiving the dimensions of
>  the image directly instead of the "vc" points (since, as I said, it is
>  in this case where speed really becomes an issue). It is doing
>  additions rather than multiplications, which appears to work faster.
>
>  FUNCTION rc_fromimage, v0,v1,v2,v3,dims
>        RETURN, REBIN(v1 # INDGEN(dims[0]), [3, dims]) + REBIN(REFORM(v2 #
>  INDGEN(dims[1]), 3, 1, dims[1], 1), [3, dims]) + REBIN(REFORM(v3 #
>  INDGEN(dims[2]), 3, 1, 1, dims[2]), [3, dims])
>  END
>
>  within testspeed, the tests looks like "rc =
>  rc_fromimage(v0,v1,v2,v3,dims)"
```

Sorry, small correction adding v0 which I forgot before:

```
FUNCTION rc_fromdims, v0,v1,v2,v3,dims
 RETURN, $
  REBIN(v0, [3, dims], /SAMPLE) + $
  REBIN(v1 # INDGEN(dims[0]), [3, dims], /SAMPLE) + $
  REBIN(REFORM(v2 # INDGEN(dims[1]), 3, 1, dims[1], 1), [3, dims], /
SAMPLE) + $
  REBIN(REFORM(v3 # INDGEN(dims[2]), 3, 1, 1, dims[2]), [3, dims], /
SAMPLE)
END
```

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by pgrigis on Thu, 30 Sep 2010 14:59:59 GMT

View Forum Message <> Reply to Message

[skip]

```
>
>  FUNCTION vc2rc_accel, v0,v1,v2,v3,vc
>          npoints = (SIZE(vc, /DIMENSIONS))[1]
>          for i=0L, npoints-1 DO BEGIN
>                  vc[*,i] = vc[0,i] * v1 + vc[1,i] * v2 + vc[2,i] * v3 + v0
>          endfor
>          RETURN, vc
>  END
```

No, that's using a for loop - that's why it is slow.
You want something like this (no loops):

```
vc0=vc[0,*]
vc1=vc[1,*]
vc2=vc[2,*]
vc[0,*]=vc0*v1[0]+vc1*v2[0]+vc3*v3[0]
vc[1,*]=vc0*v1[1]+vc1*v2[1]+...
vc[2,*]=vc0*v1[2]+...
```

Ciao,
Paolo

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by pgrigis on Thu, 30 Sep 2010 15:03:43 GMT

View Forum Message <> Reply to Message

On Sep 30, 10:59 am, Paolo <pgri...@gmail.com> wrote:
> [skip]
>
>

>
>> FUNCTION vc2rc_accel, v0,v1,v2,v3,vc
>> npoints = (SIZE(vc, /DIMENSIONS))[1]
>> for i=0L, npoints-1 DO BEGIN
>> vc[*,i] = vc[0,i] * v1 + vc[1,i] * v2 + vc[2,i] * v3 + v0
>> endfor
>> RETURN, vc
>> END
>
> No, that's using a for loop - that's why it is slow.
> You want something like this (no loops):
>
> vc0=vc[0,*]
> vc1=vc[1,*]
> vc2=vc[2,*]
> vc[0,*]=vc0*v1[0]+vc1*v2[0]+vc3*v3[0]
> vc[1,*]=vc0*v1[1]+vc1*v2[1]+...
> vc[2,*]=vc0*v1[2]+...
>
> Ciao,
> Paolo

well, there's a bit of an index mismatch
in there...

vc[0,*]=vc0*v1[0]+vc1*v2[0]+vc2*v3[0]
vc[1,*]=vc0*v1[1]+vc1*v2[1]+vc2*v3[1]
vc[2,*]=vc0*v1[2]+vc1*v2[2]+vc2*v3[2]

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Thu, 30 Sep 2010 16:38:33 GMT
View Forum Message <> Reply to Message

On Sep 30, 5:03 pm, Paolo <pgri...@gmail.com> wrote:
> On Sep 30, 10:59 am, Paolo <pgri...@gmail.com> wrote:
>
>
>
>> [skip]
>
>>> FUNCTION vc2rc_accel, v0,v1,v2,v3,vc
>>> npoints = (SIZE(vc, /DIMENSIONS))[1]
>>> for i=0L, npoints-1 DO BEGIN
>>> vc[*,i] = vc[0,i] * v1 + vc[1,i] * v2 + vc[2,i] * v3 + v0
>>> endfor
>>> RETURN, vc

>>> END
>
>> No, that's using a for loop - that's why it is slow.
>> You want something like this (no loops):
>
>> vc0=vc[0,*]
>> vc1=vc[1,*]
>> vc2=vc[2,*]
>> vc[0,*]=vc0*v1[0]+vc1*v2[0]+vc3*v3[0]
>> vc[1,*]=vc0*v1[1]+vc1*v2[1]+...
>> vc[2,*]=vc0*v1[2]+...
>
>> Ciao,
>> Paolo
>
> well, there's a bit of an index mismatch
> in there...
>
> vc[0,*]=vc0*v1[0]+vc1*v2[0]+vc2*v3[0]
> vc[1,*]=vc0*v1[1]+vc1*v2[1]+vc2*v3[1]
> vc[2,*]=vc0*v1[2]+vc1*v2[2]+vc2*v3[2]

Got it, thanks. I changed it in my test, it turned out to be slower than the original version (about 30% slower).

By now, seeing no way to improve the original version with an arbitrary vc, I stick to the version of "adding REBINS" for the case where all indexes from an image are given as input. This is the critical case for my application anyways. When a smaller region is wanted, the original version can be used.