

---

Subject: Re: IDL calls C calls IDL?

Posted by [hotplainrice@gmail.co](mailto:hotplainrice@gmail.co) on Sat, 13 Sep 2008 01:42:05 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Sep 11, 10:42 pm, Joost Aan de Brugh <joost...@gmail.com> wrote:

> On Sep 11, 11:53 am, "hotplainr...@gmail.com" <hotplainr...@gmail.com>  
> wrote:

>

>

>

>> Hey again,

>

>> I've been thinking, I have code that does

>

>> LINKIMAGE 'func' ,.....

>

>> for t=0,tmax-1

>> func(data)

>> idl\_function(data)

>> endfor

>

>> Is it possible to call idl\_function from within C ?

>

>> I want it to be like this

>

>> In IDL:

>> func(data)

>

>> Then it passes onto C which does

>> for (t=0; t<tmax;t++)

>> {

>> do whatever it does with data

>> CALL IDL\_FUNCTION(data)

>

>> }

>

>> This eliminates me calling func a few million times

>

>> Can it be done?

>

> Hello,

>

> And in the above example, IDL takes the initiative, calls func (which  
> is, if I am right, a C function) and does its own IDL function. (IDL  
> calls C a million times).

>

> Below, C takes the initiative, creates a forloop, and in each  
> iteration it does something with data and then has IDL do something

> with data by calling IDL\_FUNCTION. (C calls IDL a million times).  
>  
> Which of them is less nasty is not easy to say. To figure out how to  
> call IDL from C is better known by C-people. So that could better be  
> asked in C-groups.  
>  
> I do not know what your functions do with the data. But if the order  
> is not so important, you could try to do something like.  
>  
> In C:  
> (t=0; t<tmax;t++)  
> {  
> C-code(data);  
>  
> }  
>  
> and in IDL:  
>  
> Call\_C\_Code(data)  
>  
> ; Now it's IDL time.  
> for t=0,tmax-1 do IDL\_FUNCTION,data  
>  
> But here, C does its whole task before IDL starts. This may be a  
> problem depending on the actions. It may be possible that C can  
> generate the actions that IDL can perform. For example:  
>  
> In C:  
>  
> (t=0; t<tmax;t++)  
> {  
> C-code(data,actions); // Actions is an output by C passed back to  
> IDL. Data is left unharmed.  
>  
> }  
>  
> in IDL:  
>  
> Call\_C\_Code(data,actions)  
>  
> ; Now it's IDL time.  
> for t=0,tmax-1 do begin  
> IDL\_PERFORM\_ACTION,actions,data  
> IDL\_FUNCTION,data  
> End  
>  
> Best regards,  
> Joost

I should have phrased it in a better way.

It is like this

It all starts from within IDL

So IDL does

```
for t=0,tmax-1 do begin
  Part 1 of code written in IDL
  Part 2 of code written in IDL
  Part 3 of code written in IDL
end
```

Then I managed to convert part 1 and 3 into C and CUDA which is leaps ahead faster.

So now it looks like this

```
LINKIMAGE, 'Part1_C', 'file', 1, 'call_cuda'
LINKIMAGE, 'Part3_C', 'file', 1, 'c_part3'
```

```
for t=0,tmax-1 do begin
  Part 1 of code written in C that calls CUDA
  Part 2 of code written in IDL
  Part 3 of code written in C
end
```

Then I managed to combine the both Part 1 & 3

```
LINKIMAGE, 'Part1+3_C', 'file', 1, 'call_cuda'
```

```
for t=0,tmax-1 do begin
  Part (1+3) of code written in C & CUDA
  Part 2 of code written in IDL
end
```

Okay, so this is the nasty bit. Everytime I call Part(1+3), it calls the C function. There is no memory transfer because they share memory addresses, which is all great. However, that C function will allocate memory and transfer data to the graphics card because it is a normal CUDA procedure, allocate and transfer then execute computation kernel.

The issue is it has to allocate and transfer data to the graphics card. It takes awhile especially when you have 434 MBs to send every

iteration. That could easily be  $x4 = 1.5$  Gigs of data to the GPU.  
Whats worse is that it sends the same data all the time.

So imagine  $t_{max}=10^8$ , that is  $10^8 * (\text{the time to transfer to and back}) = 10^8 * 0.01 = 10^6$  seconds.

My aim is for

```
LINKIMAGE, 'fast_C', 'file', 1, 'call_cuda'
```

Within IDL

```
<some IDL commands>  
fast_C(..., ..., ...)  
<some IDL commands>
```

Within the C code,

```
Call_CUDA  
end of file
```

Within CUDA code,  
Allocate memory and transfer data

```
for (t=0; t<tmax;t++)  
{  
  Part 1  
  Part 3      // They don't necessarily have to be in order  
  Transfer data from GPU to host  
  CALL_IDL ( Part 2)  
}
```

So I essentially have eliminated transferring to the GPU  $10^8$  times and thus a significant fraction of  $10^6$  seconds.

Why don't I convert Part 2?

I think its too hard for me to re-invent this wheel, its image writing code and some special code that is written by someone else