Subject: Re: IDL Matrix Multiply and Dual-Core CPUs
Posted by Bringfried Stecklum on Fri, 09 May 2008 16:09:57 GMT
View Forum Message <> Reply to Message

s.haenger@gmail.com wrote:
> Hi,
>
> I have a Problem with IDL 7.0
> We have to multiply large matrices. With some matrix sizes, the CPU
> usage is 100% but for most of the matrices it is 50%. (I'm runnning it
> on a Intel T7250 (Dual Core, 2GHz, 2MB L2 Cache))
>
>
> The CPU System Variable is configured like this:
> IDL> print, !CPU
> {   0   0   2   2
> 100000   0}
>
>
> Now we do this:
> matA = randomn(42, 2000, 2200)
> matB = randomn(43, 2020, 2000)
> matIdl = matA##matB
>
> So now i've got a CPU usage of 100%
>
> but with this:
> matA = randomn(42, 2500, 2500)
> matB = randomn(43, 2520, 2500)
> matIdl = matA##matB
>
> the cpu usage is around 50%-60%
>
> I've already tried to increase the TPOOL_NTHREADS and to decrease the
> TPOOL_MIN_ELTS! It didn't help!
>
> We throught it could be because the size (2500*2520=) produces an
> overflow and the matrix size gets too small or negative, so IDL uses
> just 1 thread to compute.
>
> Does anybody know how I can fix that problem?
>
>
> Thanks a lot
> Samuel
If the physical memory of your machine is not big enough to hold all arrays
the system kernel starts to swap which eats CPU.

Regards,

B. Stecklum

Subject: Re: IDL Matrix Multiply and Dual-Core CPUs
Posted by Foldy Lajos on Fri, 09 May 2008 18:16:12 GMT
View Forum Message <> Reply to Message

I have run some tests on a quad-core Intel Core2 Q6600 / linux 64 bit
machine.

On Fri, 9 May 2008, s.haenger@gmail.com wrote:

```
> Hi,
>
> I have a Problem with IDL 7.0
> We have to multiply large matrices. With some matrix sizes, the CPU
> usage is 100% but for most of the matrices it is 50%. (I'm runnning it
> on a Intel T7250 (Dual Core, 2GHz, 2MB L2 Cache))
>
>
> The CPU System Variable is configured like this:
> IDL> print, !CPU
> {        0        0        2        2
> 100000        0}
>
>
> Now we do this:
> matA = randomn(42, 2000, 2200)
> matB = randomn(43, 2020, 2000)
> matIdl = matA##matB
>
> So now i've got a CPU usage of 100%
>
```

```
# of threads    IDL 7 time
      1        12.476210
      2         6.5931890
      3         5.2085290
      4         4.9191489
```

it scales well for two cores, so the CPU usage should be near 100% for two
threads.

```
> but with this:
> matA = randomn(42, 2500, 2500)
> matB = randomn(43, 2520, 2500)
```

> matIdl = matA##matB
>
> the cpu usage is around 50%-60%

```
    # of threads    IDL 7 time
          1        22.034877
          2        11.681226
          3         9.7771089
          4         9.3093379
```

again, CPU usage should be near 100% for two cores.

Just for comparison, ATLAS (http://math-atlas.sf.net) times:

```
    # of threads    IDL 7 time
          1         4.4285851
          4         1.1784132
```

and

```
    # of threads    IDL 7 time
          1         7.8148808
          4         2.1345751
```

regards,
lajos

Subject: Re: IDL Matrix Multiply and Dual-Core CPUs
Posted by Foldy Lajos on Fri, 09 May 2008 18:20:09 GMT

View Forum Message <> Reply to Message

oops, I have written IDL 7 time to the ATLAS test. Corrected below.

lajos

I have run some tests on a quad-core Intel Core2 Q6600 / linux 64 bit
machine.

On Fri, 9 May 2008, s.haenger@gmail.com wrote:

  Hi,

  I have a Problem with IDL 7.0
  We have to multiply large matrices. With some matrix sizes, the CPU

usage is 100% but for most of the matrices it is 50%. (I'm runnning it
on a Intel T7250 (Dual Core, 2GHz, 2MB L2 Cache))

The CPU System Variable is configured like this:

```
IDL> print, !CPU
{         0         0         2         2
100000         0}
```

Now we do this:

```
matA = randomn(42, 2000, 2200)
matB = randomn(43, 2020, 2000)
matIdl = matA##matB
```

So now i've got a CPU usage of 100%

```
 # of threads    IDL 7 time
      1         12.476210
      2          6.5931890
      3          5.2085290
      4          4.9191489
```

it scales well for two cores, so the CPU usage should be near 100% for two
threads.

but with this:

```
matA = randomn(42, 2500, 2500)
matB = randomn(43, 2520, 2500)
matIdl = matA##matB
```

the cpu usage is around 50%-60%

```
 # of threads    IDL 7 time
      1        22.034877
      2        11.681226
      3         9.7771089
      4         9.3093379
```

again, CPU usage should be near 100% for two cores.

Just for comparison, ATLAS (http://math-atlas.sf.net) times:

```
 # of threads    ATLAS time
      1         4.4285851
      4         1.1784132
```

and

```
  # of threads    ATLAS time
        1         7.8148808
        4         2.1345751
```

regards,
lajos
>

Subject: Re: IDL Matrix Multiply and Dual-Core CPUs
Posted by s.haenger on Fri, 09 May 2008 18:28:58 GMT
View Forum Message <> Reply to Message

```
On 9 Mai, 20:20, FÖLDY Lajos <fo...@rmki.kfki.hu> wrote:
>  oops, I have written IDL 7 time to the ATLAS test. Corrected below.
>
>  lajos
>
>  I have run some tests on a quad-core Intel Core2 Q6600 / linux 64 bit
>  machine.
>
>  On Fri, 9 May 2008, s.haen...@gmail.com wrote:
>
>    Hi,
>
>    I have a Problem with IDL 7.0
>    We have to multiply large matrices. With some matrix sizes, the CPU
>    usage is 100% but for most of the matrices it is 50%. (I'm runnning it
>    on a Intel T7250 (Dual Core, 2GHz, 2MB L2 Cache))
>
>    The CPU System Variable is configured like this:
>    IDL> print, !CPU
>    {           0           0           2           2
>    100000           0}
>
>    Now we do this:
>    matA = randomn(42, 2000, 2200)
>    matB = randomn(43, 2020, 2000)
>    matIdl = matA##matB
>
>    So now i've got a CPU usage of 100%
>
>     # of threads    IDL 7 time
>           1         12.476210
>           2          6.5931890
>          3          5.2085290
>          4          4.9191489
>
>  it scales well for two cores, so the CPU usage should be near 100% for two
>  threads.
>
>    but with this:
>    matA = randomn(42, 2500, 2500)
>    matB = randomn(43, 2520, 2500)
>    matIdl = matA##matB
>
>    the cpu usage is around 50%-60%
>
>     # of threads    IDL 7 time
>          1        22.034877
>          2        11.681226
>          3         9.7771089
>          4         9.3093379
>
>  again, CPU usage should be near 100% for two cores.
>
>  Just for comparison, ATLAS (http://math-atlas.sf.net) times:
>
>     # of threads    ATLAS time
>          1         4.4285851
>          4         1.1784132
>
>  and
>
>     # of threads    ATLAS time
>          1         7.8148808
>          4         2.1345751
>
>  regards,
>  lajos
>
>
```

Sorry, I forgot to mention... I'm running Windows XP 32bit with 2GB of
Ram
I also tested it on a second machine with a 3GHz Dual Core and it
showed the same cpu usages...

regards,
Samueö

Subject: Re: IDL Matrix Multiply and Dual-Core CPUs
Posted by Pierre V. on Fri, 09 May 2008 18:34:32 GMT
View Forum Message <> Reply to Message

Hi Samuel,

I saw a very similar problem with my quad-core PC running XP (32 bit)
with 4gigs of ram. I re-ran my test script on our two-core, 4-gig
linux box and got similar results with just slightly different array
sizes. Here is the script I ran:

```
cpu, /reset

help, !cpu, /str

Nk = 258
K = fltarr(Nk, Nk)

;
; Case 1.
;
Npix = 129047
d = fltarr(Npix, Nk)
t0 = systime(1)

d #= K

t1 = systime(1) - t0

print, 'Case #1: ', Npix,  t1

;
; Case 2.
;
Npix = Npix + 1
d = fltarr(Npix, Nk)
t0 = systime(1)

d #= K

t2 = systime(1) - t0

print, 'Case #2: ', Npix,  t2
```

On each of our computers case #2 used all available cores while case
#1 only used one core. The only difference between them is the
dimension of one of the arrays (Npix) is simply incremented by one.
The total memory used by the IDL process during this test is never
more and two-hundred megs or so. There is no way this problem is due
to a lack of physical memory. The sizes of these arrays are also
significantly larger then the default minimum number of elements
(default = 10000) required to enable multi-threading.

Any ideas?
Pierre

Subject: Re: IDL Matrix Multiply and Dual-Core CPUs
Posted by s.haenger on Tue, 20 May 2008 07:21:42 GMT

View Forum Message <> Reply to Message

On 9 Mai, 20:34, Pierre <pierre.villene...@gmail.com> wrote:
```
>  Hi Samuel,
>
>  I saw a very similar problem with my quad-core PC running XP (32 bit)
>  with 4gigs of ram.  I re-ran my test script on our two-core, 4-gig
>  linux box and got similar results with just slightly different array
>  sizes.  Here is the script I ran:
>
>  cpu, /reset
>
>  help, !cpu, /str
>
>  Nk = 258
>  K = fltarr(Nk, Nk)
>
>  ;
>  ; Case 1.
>  ;
>  Npix = 129047
>  d = fltarr(Npix, Nk)
>  t0 = systime(1)
>
>  d #= K
>
>  t1 = systime(1) - t0
>
>  print, 'Case #1: ', Npix,  t1
>
>  ;
```

> ; Case 2.
> ;
> Npix = Npix + 1
> d = fltarr(Npix, Nk)
> t0 = systime(1)
>
> d #= K
>
> t2 = systime(1) - t0
>
> print, 'Case #2: ', Npix, t2
>
> On each of our computers case #2 used all available cores while case
> #1 only used one core. The only difference between them is the
> dimension of one of the arrays (Npix) is simply incremented by one.
> The total memory used by the IDL process during this test is never
> more and two-hundred megs or so. There is no way this problem is due
> to a lack of physical memory. The sizes of these arrays are also
> significantly larger then the default minimum number of elements
> (default = 10000) required to enable multi-threading.
>
> Any ideas?
> Pierre

It's not a Windows Problem. We have the same Problem also with
Ubuntu...

Subject: Re: IDL Matrix Multiply and Dual-Core CPUs
Posted by Karl[1] on Tue, 20 May 2008 18:19:36 GMT
View Forum Message <> Reply to Message

On May 20, 1:21 am, s.haen...@gmail.com wrote:
> On 9 Mai, 20:34, Pierre <pierre.villene...@gmail.com> wrote:
>
>> Hi Samuel,
>
>> I saw a very similar problem with my quad-core PC running XP (32 bit)
>> with 4gigs of ram. I re-ran my test script on our two-core, 4-gig
>> linux box and got similar results with just slightly different array
>> sizes. Here is the script I ran:
>
>> cpu, /reset
>
>> help, !cpu, /str
>
>> Nk = 258
>> K = fltarr(Nk, Nk)

```
>
>>  ;
>>  ; Case 1.
>>  ;
>>  Npix = 129047
>>  d = fltarr(Npix, Nk)
>>  t0 = systime(1)
>
>>  d #= K
>
>>  t1 = systime(1) - t0
>
>>  print, 'Case #1: ', Npix,  t1
>
>>  ;
>>  ; Case 2.
>>  ;
>>  Npix = Npix + 1
>>  d = fltarr(Npix, Nk)
>>  t0 = systime(1)
>
>>  d #= K
>
>>  t2 = systime(1) - t0
>
>>  print, 'Case #2: ', Npix,  t2
>
>>  On each of our computers case #2 used all available cores while case
>>  #1 only used one core.  The only difference between them is the
>>  dimension of one of the arrays (Npix) is simply incremented by one.
>>  The total memory used by the IDL process during this test is never
>>  more and two-hundred megs or so.  There is no way this problem is due
>>  to a lack of physical memory.  The sizes of these arrays are also
>>  significantly larger then the default minimum number of elements
>>  (default = 10000) required to enable multi-threading.
>
>>  Any ideas?
>>  Pierre
>
>  It's not a Windows Problem. We have the same Problem also with
>  Ubuntu...
```

There's a lot of speculation to follow, so be warned.

Making sure that using multiple threads is really faster isn't very straightforward. There's lot of overhead involved when splitting the problem into threads. There is a lot of data movement, creating

tasks, waiting for them all to complete, etc. There are also other
factors such as memory page sizes, cache lines, etc. So, using
multiple threads isn't always a win, as hinted by the minimum data
size.

I would suppose that there is a set of "heuristics" that are used to
decide whether to multi-thread or not, based on the data size, shape,
layout and the algorithm being implemented. I wasn't very closely
involved, but when this was being developed, there were some very
interesting surprises about what sorts of problems multi-threading
would yield a net gain and what sort of problems ended up being a net
loss.

There's probably a lot of effort being made to avoid ending up with a
slower result when using multiple threads. It might be too
conservative, or the decision might be wrong due to a bug. But it
might even be correct and that changing the data size that least
little bit in this example ends up changing the decision as to whether
to use multiple threads or not.   I find it odd, given the data in the
example, but it is possible.

The only way you'll know is to ask ITTVIS why MT was rejected for one
array and not for the other.

Karl

Subject: Re: IDL Matrix Multiply and Dual-Core CPUs
Posted by s.haenger on Thu, 17 Jul 2008 08:42:21 GMT

View Forum Message <> Reply to Message

On 20 Mai, 20:19, Karl <Karl.W.Schu...@gmail.com> wrote:
> On May 20, 1:21 am, s.haen...@gmail.com wrote:
>
>
>
>> On 9 Mai, 20:34, Pierre <pierre.villene...@gmail.com> wrote:
>
>>> Hi Samuel,
>
>>> I saw a very similar problem with my quad-corePC running XP (32 bit)
>>> with 4gigs of ram. I re-ran my test script on our two-core, 4-gig
>>> linux box and got similar results with just slightly different array
>>> sizes. Here is the script I ran:
>
>>> cpu, /reset
>
>>> help, !cpu, /str

```
>
>>>  Nk = 258
>>>  K = fltarr(Nk, Nk)
>
>>>  ;
>>>  ; Case 1.
>>>  ;
>>>  Npix = 129047
>>>  d = fltarr(Npix, Nk)
>>>  t0 = systime(1)
>
>>>  d #= K
>
>>>  t1 = systime(1) - t0
>
>>>  print, 'Case #1: ', Npix,  t1
>
>>>  ;
>>>  ; Case 2.
>>>  ;
>>>  Npix = Npix + 1
>>>  d = fltarr(Npix, Nk)
>>>  t0 = systime(1)
>
>>>  d #= K
>
>>>  t2 = systime(1) - t0
>
>>>  print, 'Case #2: ', Npix,  t2
>
>>>  On each of our computers case #2 used all available cores while case
>>>  #1 only used onecore.  The only difference between them is the
>>>  dimension of one of the arrays (Npix) is simply incremented by one.
>>>  The total memory used by theIDLprocess during this test is never
>>>  more and two-hundred megs or so.  There is no way this problem is due
>>>  to a lack of physical memory.  The sizes of these arrays are also
>>>  significantly larger then the default minimum number of elements
>>>  (default = 10000) required to enable multi-threading.
>
>>>  Any ideas?
>>>  Pierre
>
>>  It's not a Windows Problem. We have the same Problem also with
>>  Ubuntu...
>
>  There's a lot of speculation to follow, so be warned.
>
>  Making sure that using multiple threads is really faster isn't very
```

> straightforward. There's lot of overhead involved when splitting the
> problem into threads. There is a lot of data movement, creating
> tasks, waiting for them all to complete, etc. There are also other
> factors such as memory page sizes, cache lines, etc. So, using
> multiple threads isn't always a win, as hinted by the minimum data
> size.
>
> I would suppose that there is a set of "heuristics" that are used to
> decide whether to multi-thread or not, based on the data size, shape,
> layout and the algorithm being implemented. I wasn't very closely
> involved, but when this was being developed, there were some very
> interesting surprises about what sorts of problems multi-threading
> would yield a net gain and what sort of problems ended up being a net
> loss.
>
> There's probably a lot of effort being made to avoid ending up with a
> slower result when using multiple threads. It might be too
> conservative, or the decision might be wrong due to a bug. But it
> might even be correct and that changing the data size that least
> little bit in this example ends up changing the decision as to whether
> to use multiple threads or not.   I find it odd, given the data in the
> example, but it is possible.
>
> The only way you'll know is to ask ITTVIS why MT was rejected for one
> array and not for the other.
>
> Karl

Thanks a lot for the response.
Actually we did ask the ITTVIS Support guys and they responded pretty
much the same things as you (except for the bug thing) :-)
I think we have to accept, that IDL just uses MT when it wants to :-)

Samuel