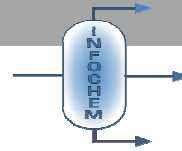




NextGen Performance[®]



A quick note on threading


Jasper van Baten, AmsterCHEM
Richard Szczepanski, KBC/Infochem

September 2013


© 2011 KBC Advanced Technologies plc. All Rights Reserved.

September 2013

The CAPE-OPEN DLL for Multiflash has been working satisfactory already for several years. In most applications that is. Two small adjustments have been made this year. This short note will elaborate on the adjustments made with respect to multi-threading support.



Infochem



Changes in the Multiflash CAPE-OPEN DLL, 2013

- Enhanced multithreading support
 - Inspired by some interesting findings at BASF
- x64 support
 - Inspired by x64 support of the Multiflash core


September 2013

CO Meeting


2

Multi-threading performance issues were flagged by BASF where equation oriented simulations are solved in a multi-threaded fashion. Both the findings and the solutions are the subject of this presentation.

In addition, an x64 version of the CAPE-OPEN DLL was produced. This was inspired by an x64 release of the core of Multiflash itself. The reason for this is simple: 64 bit processes execute more efficiently on 64 bit computers, and you cannot use 32 bit DLLs in such processes. No need to expand on this subject further.



Infochem



NextGen Performance

Two ways of utilizing all nodes on a given computer:

- Multithreading
 - Thread safety issues
 - Performance bottlenecks
- Parallel processing
 - Separate process spaces, all works well
 - Inter-process communication may be limiting


September 2013

CO Meeting


3

Modern computers are equipped with multiple cores. Development has reached a point where cores themselves do not really get faster, so if you want to exploit the capabilities of modern computers, you have to use multiple cores. There are generally two techniques: multi-threading within a single process, or parallel processing that runs multiple processes. The latter is rather simple from a CAPE-OPEN point of view, as each process essentially is single threaded and nothing really bites each other. Licensing may be an issue, as each process needs to check out a license, but this is not a CAPE-OPEN problem. However, inter-process communications may be the bottle neck for performance in this case, which is why multi-threading is frequently used.

In multi-threaded applications, we require the parts of the application to be thread safe. Then we further assess where performance bottle-necks are.



KBC + Infochem



What is the big deal with multithreading?

- Many 'legacy' implementations are not thread safe
 - Common blocks
 - Global variables
 - Static variables
 - Shared resources

- But do Apartment Threaded COM objects require this?
 - Yes!

September 2013


CO Meeting

4


The reason many CAPE-OPEN implementations are not thread safe is legacy code that contains common blocks, global variables, static variables or other shared resources. In the case of Multiflash, which was written in F77, COMMON blocks are the culprit. If two threads read from and write to these blocks at the same time, the results are unpredictable and likely wrong.

Typically COM objects advertise themselves as Apartment Threaded. By contract, this means that all COM objects may only be accessed from the thread that they were created in. So do we really need to write thread-safe CAPE-OPEN objects? The answer is yes. The multi-threaded solution for using CAPE-OPEN property packages is that you create a copy of the property package in each thread. These may therefore be called only from their respective threads, that execute concurrently.

So how to go about making the Property Package thread safe?



KBC + Infochem




Three approaches:

- What was done in the past
 - Shield access to the unsafe DLL using synchronization
- What is done now:
 - Create only a single property package from each unsafe DLL
- What will be done in the future
 - Eliminate the unsafe factors from the DLL


September 2013

5

There are several techniques. Multiflash was using synchronization in the past. This was changed this year in creating a Multiflash core DLL for each property package. The ultimate solution of course is to solve the cause rather than the symptoms, and remove the COMMON blocks.



KBC + Infochem



- What was done in the past
 - Shield access to the unsafe DLL using synchronization

Existing methods:

- Mutex (Mutually Exclusive) objects
 - Can be named, therefore can work across processes
- Critical Sections
 - Local, but faster

Clearly thermodynamic calculations cannot be performed concurrently


September 2013

6


So let us have a closer look. Windows offers several synchronization mechanisms. There are the Mutex objects, which stands for Mutually Exclusive. Only one thread can 'own' such an object at the same time. Other threads that attempt to acquire such an object are put on hold until the Mutex object has been released by the thread that owns it. Such Mutex objects can have a unique name, so they can be used even to synchronize across different processes.

A simpler and cheaper, but local, equivalent was used in Multiflash: a CriticalSection object. This was used to shield concurrent access to the core DLL from two threads. Effectively this allows only one thread to execute any thermodynamic method at the same time. Other threads wait until the thermodynamic routines become available. If significant time of each thread is spent in thermodynamic calculations, of course not much is gained by multi-threading, as most threads are waiting most of the time.

Still this solution was better than no synchronization, as that leads to unstable and wrong operation, and even crashes.



KBC + Infochem



- What is done now:
 - Create only a single property package from each unsafe DLL

No 2 DLLs with the same name can be loaded.

Create a copy of the DLL in the temp folder with unique name.

Dynamic linking: LoadLibrary, GetProcAddress

Initial delay due to loading DLL. Bigger memory footprint


September 2013

7


Another commonly used technique is to avoid the problem of COMMON blocks by creating a copy of the COMMON blocks for each instance that uses them, e.g. each property package. Of course there is only one memory location associated with a COMMON block in the module they are implemented in, which is the Infochem core DLL. So this solution implies loading the core DLL multiple times. Windows prevents loading DLLs with the same name more than once, so part of this method involves giving the DLL a unique name, and copying it to the temporary folder. Static linking can then no longer be done, so LoadLibrary and GetProcAddress are used to access the functionality of the DLLs.

Although this solution is much better, as two threads can now access thermodynamic calculations concurrently, each using their own copy of the DLL, it is not perfect. Many copies of the DLL leads to a bigger memory footprint. Also there is some more overhead involved with creating a property package. Copying the DLL, checking out a license, etc. Some time here can be saved by re-using DLLs as property packages get destroyed and created.

This solution works well. As a side effect, some other issues were solved that resulted of too many Property Packages being created from one DLL. Now each DLL creates only one Property Package.



KBC + Infochem



- What will be done in the future
 - Eliminate the unsafe factors from the DLL



This requires removing all COMMON blocks. F77 not very suitable. A much bigger project.

September 2013

8

The ultimate solution of course is to cure the problem, not the symptoms. The COMMON blocks need to be removed. This is a much bigger project, as this requires recoding at least parts of the Multiflash core.

This is in the pipe line, as this is required in non CAPE-OPEN contexts already.



What cannot be done?

Heap memory allocations are synchronized.

No two threads can simultaneously allocate memory.


This impacts multithreading performance.

As long as COM is used, this cannot be helped. New object model should take this into account.


September 2013 9

So – we are in a much better state already. Are we there yet? Alas, no.

We got rid of the synchronization in the multiflash CAPE-OPEN wrapper, but heap memory allocations are also synchronized. The memory that is suspected to cause the performance impact here is the in- and output arguments to the CAPE-OPEN routines, particularly the BSTR and SafeArray values passed around. These are allocated by the PME and released by the PMC, and vice versa. Hence, we cannot prevent that. As long as COM is used that is. Another object model is the only way out here.



KBC + Infochem



Conclusions

- Multi-threading improvements are customer driven
- The current implementation is intermediate
- The final answer is thread safe core libraries

- x64 is now supported

September 2013

10

To conclude:

We improved our multi-thread handling. Not because this is the right thing to do, not because it is cool, but because our customers require us to do so.

The current implementation is satisfactory for now, but not the final answer. The final answer is in the pipe line though.

64 bit operation is now also supported. It is remarkable that most CAPE-OPEN application are still 32-bit at this point in time. Although customer demand at this point was present but small, we foresee that PMC vendors have little choice but to be both x64 and multi-thread compliant in the near future, to keep up with modern computing. PMEs have more control over whether multi-threading, or perhaps parallel processing, is used....