

Re: GC performance – GC fragility

Source:

<http://coding.derkeiler.com/Archive/Delphi/borland.public.delphi.non-technical/2008-01/msg03658.html>

- *From:* Andre Kaufmann <andre.kaufmann_re_move@xxxxxxxxxxxxx>
 - *Date:* Mon, 28 Jan 2008 20:01:43 +0100
-

Eric Grange wrote:

You normally will end with a heap looking like a swiss cheese.

It won't interestingly enough. Though the GC heap will look like one before compaction.

AFAIK FastMM tries to fight this with granularity and passing different allocation sizes to different "heap blocks", but it can't be IMHO generally be prevented.

Well the point you're missing here is that memory is the MM operates on virtual memory. If you only have macro-fragmentation in the virtual

The point you are missing, that it's still a linear address space, or isn't the address space each process has 2 GB linear ?

Isn't the whole process and the system using virtual memory ?

Isn't GC >not using< virtual memory ?

The only way to (somewhat) directly address and map physical memory under Windows x86 I know is using AWE.

Virtual memory only means, that physical memory is virtually mapped to a virtual memory address. But mapping >used< memory to another address location would make the application some trouble, since it's using linear addresses ? Wouldn't it ?

—> GC also uses virtual memory functions to allocate memory.

—> Linearity of GC memory is too only virtual

—> If not GC could only allocate memory in a linear free block

—> which would be quite hard to find

FastMM uses VirtualAlloc which returns a block of memory -> so what ?

If these blocks are somehow distributed in the linear address space it has no effect on the fragmentation of the

Re: GC performance – GC fragility

virtually linear heap.

It's simply a linked list of memory blocks of a specified granularity in a linear address space.

If you have 1 byte of application heap allocated in each of 1000 virtual allocate blocks of 64K they still would be allocated and couldn't be freed -> heap fragmentation.

memory space, you can operate with limited fragmentation-induced waste (ie. below a constant amount) in the allocated memory space.

In practice you're looking at negligible amounts of memory waste

I don't want to argue that fragmentation is general a problem in practice, but I wouldn't call it negligible. Depending on the allocation scheme. But this is also true for GC. You can always find examples where the one or the other heap wins. When managed applications are in practice slower than native ones it's not >only< GCs fault.

compared to a GC (as all test cases so far have illustrated).

Well my little pseudo code still works in BDS 2006 – at least I remember to have it tested there. I'll give it a try tomorrow in my office, since I haven't it yet installed at home -> don't want .NET 1.1 SDK.

The only fragmentation that remains in FastMM is that of virtual memory when allocating large blocks. Though in 64bit, with virtually unlimited

So why does FastMM distinguish between different allocation sizes – IIRC 2 or 3 -> because of fragmentation ?

virtual memory that wouldn't be a problem at all (and in 64bit there are virtual memory solutions to reallocating large blocks to arbitrary sizes without a copy).

You can't map memory on your own in Windows. You can only reserve memory and then later commit it for usage.

The point is even in 64 bit you can't simply reserve the whole 64 bit address range, since mapping tables are limited too.

It's a big problem of long running applications.

FastMM and NexusDB (which uses a similar tactic) have been used in long running servers, and they are known to work without any stalls... unlike

Re: GC performance – GC fragility

It depends on the allocation scheme. If the heap is fragmented, it doesn't mean that the applications has stalls or runs out of memory. Simply it's using somewhat more memory.

GC-based alternatives, where the GC stability is so poor that MS doesn't

Any links ?

recommand the use of the concurrent GC in servers, but use of the blocking GC to maintain stability... that's because the concurrent GC can be gotten into a cycle of never ending GC compaction if your allocation rate exceeds or equals the compaction rate, a condition easy to reach when part of your allocated pool is swapped out.

And I can easily blow off a native heap by fragmenting it.

how a native heap can (generally) prevent fragmentation,

> without compacting the memory ?

I suggest looking up the basm ng archive in google, there were quite a few discussions on the subject (look for memory manager challenge threads).

If I allocate memory:

2 Bytes

3 Bytes

4 Bytes

6 Bytes

In a virtual block of memory and free the 3 bytes and 4 bytes and want to allocate 8 bytes, the hole inside this "virtual memory" block can't be used or filled by a native heap. No way.

Eric