USA (English) Copy Code

Coding for Performance: Data alignment and structures

By Sumedh Naik , published on September 26 , 2013

This article collects the general knowledge and Best-Known-Methods (BKMs) for aligning of data within structures in order to achieve optimal performance.

Every data type has an alignment associated with it which is mandated by the processor architecture rather than the language itself. Aligning data elements allows the processor to fetch data from memory in an efficient manner and thereby improves performance. The compiler tries to maintain these alignments for data elements to provide optimal performance. The typical alignment requirements for data types on 32-bit and 64-bit Linux* systems as used by the Intel® C++ Compiler are shown below:

Data Type	**32-bit (bytes)**	**64-bit (bytes)**
char	1	1
short	2	2
int	4	4
long	8	8
float	4	4
double	8	8
long long	8	8

long double	4	16
Any pointer	4	8

Table1: typical alignment requirements for data types on 32-bit and 64-bit Linux* systems as used by the Intel® C++ Compiler

In general, the compiler will try to fulfill these alignment requirements for data elements whenever possible. In the case of the Intel® C++ and Fortran compilers, you can enforce or disable natural alignment using the *–align* (C/C++

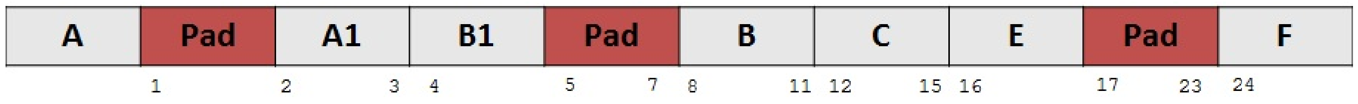
, Fortran

) compiler switch. For structures that generally contain data elements of different types, the compiler tries to maintain proper alignment of data elements by inserting unused memory between elements. This technique is known as 'Padding'. Also, the compiler aligns the entire structure to its most strictly aligned member. The compiler may also increase the size of structure, if necessary, to make it a multiple of the alignment by adding padding at the end of the structure. This is known as 'Tail Padding'. Thus padding improves performance at expense of memory. In case of the Intel® Xeon Phi™ Coprocessor, where the amount of memory available to the application is limited in itself, this poses a serious problem.

BKM: Minimizing memory wastage: One can try to minimize this memory wastage by ordering the structure elements such that the widest (largest) element comes first, followed by the second widest, and so on. The following example helps illustrate the effect of ordering of data elements on the size of the structure.

```
struct s1 { char a; short a1; char b1; float b; int c; char e; double f; }; Size
of Struct s1 = 32
```

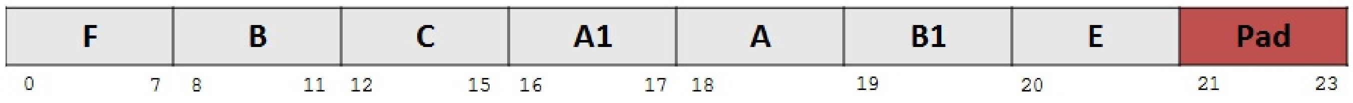
The structure s1 has 11 bytes of padding as illustrated by the diagram below:



Consider the following structure s2

```
struct s2 { double f; float b; int c; short a1; char a,b1,e; }; Size of Struct s2 = 24
```

This structure contains only 3 bytes of tail padding as indicated by the following figure and saves memory. Hence, simply re-arranging the elements during the structure definition may help in avoiding memory wastage.



BKM: Touching only some elements at a time: An exception to this ordering of elements in structures is if your structures are bigger than a cache line, which is 64 bytes in case of Intel Xeon Phi coprocessor, and some loops or kernels touch only a part of the structure. In this case, it may be beneficial to keep the parts of the structure that are touched together in memory, which may lead to improved cache locality.

BKM: Splitting larger structures: If your structures are larger than a cache line with some loops or kernels touching only a part of the structure then you may consider reorganizing the data by splitting the large structure into multiple smaller structures which are stored as separate arrays. This potentially improves the density of touched data and thereby improves cache locality.

BKM: Forcing specific alignment using align(n): You can also use the *__declpsec(align)* attribute to direct the compiler to align data more strictly than it otherwise would. The syntax for this extended-attribute is as follows:

C/C++:
__declpsec(align(n)) <data type declaration>

Fortran:
cDEC$ ATTRIBUTES ALIGN: n:: <data type declaration>

Where n is the requested alignment and is an integral power of 2, up to 4096 for the Intel C++ Compiler and up to 16384 for the Intel Fortran Compiler.

You can use this attribute to request alignments for individual variables, or structures of static or automatic storage duration. However, with that said, although you can increase the alignment of a struct, this attribute does not adjust the alignment of elements within the struct. By placing the __declspec(align) before the keyword struct, you request the appropriate alignment for only the objects of that type. Let me illustrate my point with the following example:

```
__declspec(align(32)) struct s2
{
        char a2;
        int b2;
};

struct s1
{
        char a1;
        struct s2 b1;

};
```

In the above example, the alignments for char *a2* and int *b2* still remain 1-byte and 4-bytes respectively, which are the default. However, each instance of struct *s2* is aligned to a 32 byte boundary as declared in the *__declspec*. Hence, each instance of struct *s2* present within struct *s1* will be aligned to a 32 byte boundary.

USA (English)

BKM: Alignment of dynamically allocated memory: We can further extend this example, by dynamically allocating an array of struct *s2*.

```
struct s2 *arr1;
arr1=(struct s2 *)_mm_malloc(sizeof(struct s2)*my_size,32);
```

In this case, you still need to use *_mm_malloc* or a POSIX equivalent to allocate aligned memory to the pointer, but by using the *__declspec(align(32))*, you are enforcing an alignment of 32 for each and every element of the array *arr1* (but not the internal members of the s2 struct).

BKM: Using align(n) and structures to force cache locality of small data elements: You can also use this data alignment support to advantage for optimizing cache line usage. By clustering small objects that are commonly used together into a struct, and forcing the struct to be allocated at the beginning of a cache line, you can effectively guarantee that each object is loaded into the cache as soon as any one is accessed, resulting in a significant performance benefit. For example, consider two variables *i* and *j* which are frequently accessed together but can fall on different cache lines. You can instead declare them as:

```
__declspec(align(16)) struct {int i,j} sub;
```

By declaring the variables in this fashion, the compiler ensures that the variables are allocated on the same cache line.

These Best-Known-Methods aim at influencing programmers to write efficient codes or to rework codes to improve performance. You can read more about the how the Intel C++ amd Fortran Compiler handle data alignment at https://www-ssl.intel.com/content/dam/www/public/us/en/documents/guides/itanium-software-runtime-architecture-guide.pdf

.

1

Product and Performance Information

[1] Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserverd for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Give Feedback

USA (English)

Featured Topics

Artificial Intelligence

Game Development

Internet of Things

View All Topics

Featured Tools

Intel® oneAPI Beta Toolkits(Beta)

Compilers from Intel

Performance Libraries

Intel® DevCloud

Open Source

Manage Your Tools

Product Support

Product Downloads

Product Forums

License & Renewal FAQ

Connect

Community Experts

Intel® Developer Mesh

Recent Updates

YouTube* Channel

Company Information

Our Commitment

Communities

Investor Relations

USA (English)

Contact Us

Newsroom

Jobs