



for Playstation®3

Game developers are now presented with a rapidly-growing range of different multi-core processors. All most of these processors have in common is that they are very time-consuming to program for, give very unpredictable performance, use incompatible programming models, and require a lot of time and effort to learn how to program. Offload™ for Playstation®3 has been designed with one primary goal: make it easy for game-developers to get a big performance boost with multi-core processors, but without sacrificing portability, readability, or deadlines to get there.

Offload™ has been developed with input from market leading PS3 games developers. Multi-core processors promise incredible games, with low power consumption. But they also create a huge drain on development resources and high risk of missed deadlines. Codeplay Offload reduces the development costs and risks of multi-core processors to the normal level of risk of C++ games development.

How Offload Works

Codeplay Offload™ takes in C++ source, with offload blocks marked out for call-graph-duplication and offloading to accelerator cores.

```
int ppu_function () {
    int x; // x is in shared memory
    __offload {
        int y; // y is in local memory
        y = f (&x, &y);
        /* 'f' is duplicated and called with
           a shared-memory-pointer
           (1st argument) and
           a local-memory-pointer
           (2nd argument)*/
    }
}
```

With little effort this allows the offloading of complex C++ code including virtual methods, multiple inheritance, function pointers, call-backs, etc. The system calculates the layout of the C++ data-structures and the calling conventions of the C++ functions and methods, and then outputs C code (including makefiles) for the different processors in the system with the structure layout and calling conventions hard-coded into the C. This allows different compilers, with different data-structure layouts, to compile for different processor cores, but they can safely be accessing the same data structures.

Offload™ handles GCC3, GCC4 and Microsoft C++ class layouts. The Offload™ tool also outputs C macros to handle shared memory access. In this way, the user of the system can define those macros to access shared memory in any way (dma, software cache etc)desired. This allows users to add in support for software transactional memory, for example, or data-dependency checking.

Also, the Offload™ tool can output advice to highlight places where it is not safe for the compiler to perform an optimization without the user changing the code. This allows streaming DMA, for example, to be easily inserted into existing source code.

Call-Graph Duplication

Call-graph duplication is the main hard work that the Offload compiler automatically performs behind the scenes for the programmer. Call-graph duplication enables offloading C++ code inside offload blocks without further changes to the code. (see example above). The Offload compiler separately compiles the code for the SPU and links it up with the PPU code (PPU code controls the SPU code, SPU code can reference data in host memory).

The key benefits of Call-Graph-Duplication are:

1. Reduces the amount of error-prone work required to move code onto an accelerator processor.
2. Allows programmers to keep their source code portable and easy-to-maintain.
3. Removes the requirement to hand-duplicate functions for different processor cores and memory spaces.
4. Hides the complexity of offloading complex C++ to SPU

Consider a simple example:

```
int f(int* a, int* b) {
    return *a + *b;
}
```

Without call-graph duplication, we would have to create 2 different versions of the function 'f' - for the PPU and one for the SPU. With call-graph duplication, we only need one version of function 'f' and the compiler *automatically* generates the required SPU version. This leaves the code more maintainable, easier to read, and much easier to write. It is especially useful when offloading a lot of code onto SPU.

Call-graph duplication is also useful if we decide that (perhaps for performance reasons) we want to move the variable 'x' from shared memory to local memory. Without call-graph-duplication, this would involve changing the implementation of 'f', but with call-graph-duplication, the compiler works out that 'f' needs to be compiled with different access operations for 'x'.

"PlayStation" is a registered trademark of Sony Computer Entertainment Inc.

Less code to write for the programmer

To illustrate how much less code there is to write using Offload, this is how the programmer might have to write versions of function 'f' without call-graph-duplication:

```
int f(int* a, int* b) {
    return *a + *b;
}
int spu_f(int* a, int* b) {
    return *a + *b;
}
int spu_f_local_global(int* a, __ea int* b) {
    return *a + fetch_int(b);
}
int f_global_local(__ea int* a, int* b) {
    return fetch_int(a) + *b;
}
int f_global_global(int* a, int* b) {
    return fetch_int(a) + fetch_int(b);
}
```

Overloading for Performance

If automatic duplication doesn't give you the performance you need, then you can manually override the default behaviour. For example, if you want to modify the 'f' function to use a different mechanism to read the pointer 'a' in the case where 'a' is in shared memory, you could write the following overloaded function:

```
__offload int f(int __outer * a, int* b) {
    return fetch_manual(a) + *b;
}
```

The Offload tool will choose to use this implementation of 'f' in the case where 'a' is an 'outer' (or shared memory) pointer, and use the normal implementation of 'f' when 'a' is a local pointer, or when called from PPU. This allows performance optimizers to write special high-performance versions of critical functions without having to modify the main source code, or lose source portability.

Core-Specific Optimizations

The Offload compiler implements optimizations for different types of cores. For example, casts from scalars to AltiVec vectors are performed using splat instructions.

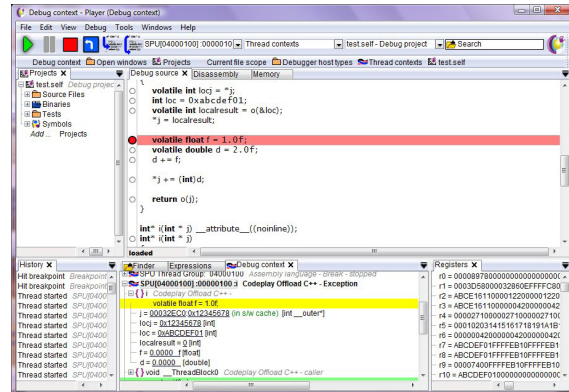
Sometimes there are more efficient intrinsics available than the corresponding GNU intrinsics and the Offload compiler can be configured to select these efficient instructions.

Useful Compiler Advice Messages

The Offload compiler implements various command options to help programmers identify performance bottlenecks in offloaded code. For example, pointer variables initialised with data outside the offload block are expensive to dereference. The option `-offloadwarnouterdeduction` makes the compiler issue a warning so that the programmer can, if possible, move the initializer data inside the offload block and the pointer dereference operation then becomes fast.

Debugging

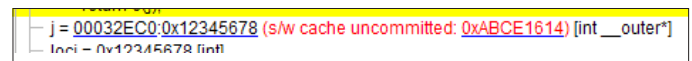
The Offload compilers output C code which can be compiled with normal compilers (and possibly with different compilers for different processor cores) and debugged with normal debuggers. However, to debug everything together, and see the impact of the software cache, developers need to use Codeplay's debugger, called Player.



Users can hover the mouse over a variable to see its value. Each processor core is shown as a normal thread with a normal call-stack. Everything behaves as users would expect for a single-core or normal multi-threaded processor, even though the system is working across PPU and SPU on the Cell processor.

The debugger handles code compiled with GCC and the standard DWARF debugger format, as well as code transformed with Codeplay's Offload™ tool.

The debugger also shows the effect of the software cache, letting programmers see the effect of writing code that runs on multiple processors:



This is showing that a value (0xABCE1614) has been written to the shared-memory pointer "j". The value is still in the software cache and not committed to shared memory, so shared memory still contains the value 0x12345678.

Try Today

Offload: Community Edition is currently free to download and use for commercial games, subject to licensing terms.

There are detailed walk-through examples showing the whole process of using Offload blocks step-by-step at <http://offload.codeplay.com>

Visit offload.codeplay.com to register an account to view forum discussion and download the latest version of the Offload System.

"PlayStation" is a registered trademark of Sony Computer Entertainment Inc.