# 1、The methodologies of software porting

## 1.1 Program execution process

Software is the set of instructions that tell the processor what to do. All software is created through the process of programming with programming languages. The programming language will be translated into machine language on the computer and executed on the processor in the form of instructions.

## 1.2 Instruction difference between POWER processor and x86 processor

The instruction set, also called ISA (instruction set architecture), is part of a computer that pertains to programming, which is more or less machine language. The instruction set provides commands to the processor, to tell it what it needs to do. The instruction sets used by processors of different architectures are difference. The POWER architecture processor uses the RISC-based (Reduced Instruction Set Computing) ISA instruction set architecture, while the x86 architecture uses the CISC-based (Complex Instruction Set Computing) instruction set architecture.

Therefore, the software under the x86 architecture must be ported before it can run on the POWER platform

## 1.3 Common issues during porting

1. Compatibility issues of make file and configure file

Makefile tell make what to do and also tells make how to compile and link a program. Building executables in different architecture require to change the variable definitions or with different compilers or compiler flags.

For example, developer requires to modify configure.guess file to support POWER architecture in python2.7.

ppc64le:Linux:\*:\*)

GUESS=powerpc64le-unknown-linux-$LIBC

;;

ppcle:Linux:\*:\*)

GUESS=powerpcle-unknown-linux-$LIBC

;;

vax:Linux:\*:\*)

GUESS=$UNAME\_MACHINE-dec-linux-$LIBC

;;

x86\_64:Linux:\*:\*)

set\_cc\_for\_build

GUESS=$UNAME\_MACHINE-pc-linux-$LIBCABI

1. Compatibility issues of assembly instructions

Each architecture has its own assembly language. Usually a Compiler can only create code for one architecture, and there might be optional flags which enable compiler to build executables on other platforms. When these flags get enabled, the program will usually only run on processors that support these extensions.

Such as the function of acquiring CPU serial number, the implementation of the two platforms is completely different. The assembly instructions of the X86 platform need to be replaced to POWER platform accordingly.

X86：

asm volatile

(

"movl $0x01, %%eax; \n\t"

"xorl %%edx, %%edx; \n\t"

"cpuid; \n\t"

"movl %%edx, %0; \n\t"

"movl %%eax, %1; \n\t"

: "=m"(s1), "=m"(s2)

);

PPC64LE：

Read from system file "/proc/device-tree/vpd/"

1. Compatibility issues of SIMD instructions

Single Instruction, Multiple Data (SIMD) units refer to hardware components that perform the same operation on multiple data operands concurrently. SIMD units have been available in most architectures. But the instruction sets provided by one architecture platform is completely different to another. For example, Intel introduced the advent of the MMX, SSE (Streaming SIMD Extensions), and AVX (Advanced Vector Extensions) ISA extensions. But IBM introduced VSX instructions.

For example, the instruction of popcnt, developers need to replace the X86 SSE instruction set with the PPC64le instruction in the code.

X86：

#define popcnt(x) asm("popcnt %[x], %[val]” : [val] "+r" (x) : : "cc")

PPC64LE:

#define popcnt(x) \_\_builtin\_popcount(x)

1. Compatibility issues of Memory page size

Physical memory partitioned into equal sized page frame. Memory only allocated in page frame sized increments. The memory page frame size used by different processor is different. The Intel processor generates a page size of 4k, and the POWER processor generates a page size of 64k by default. Allocating wrong page size in the application may cause unpredictable problems.

For example, the page size was set to 4K in mozjs17 code. It will trigger a segment fault while mozjs17 runs on platform with 64K page size. Developers need to modify the code and set the page size to 64K.

#if defined(SOLARIS) && (defined(\_\_sparc) || defined(\_\_sparcv9))

const size\_t PageShift = 13;

const size\_t ArenaShift = PageShift;

#elif defined(\_\_powerpc\_\_) || defined(\_\_aarch64\_\_)

const size\_t PageShift = 16;

# 2、The porting Process

There are four steps involved during software porting: preparing, analyzing, porting and optimizing.

## 2.1 Preparing

For preparing POWER platform environment, please choose one of the following methods:

1. Apply the test environment of migrating from the Remote Test Center of the Inspur Power System；（Contact Info：http://www.inspurpower.com/ziyuan/）
2. Local POWER servers that can connect to the Internet。

## 2.2 Analyzing

This stage is mainly to analyze the software stack and develop the porting strategy

### 2.2.1 Analyzing software stack

#### 2.2.1.1 Basic environment

This stage is mainly to analyze the software environment, including operating system, middleware, compiler, and etc. Scoping these software environments will help developers fitting these software environments wile porting software to POWER platform.

#### 2.2.1.2 Analyzing application layer

Analyzing the software packages and their dependences based on the source code. If users can obtain the source code freely, we call it open source software. If users can’t obtain the source code without proper authorization, we call it closed source software. Regarding to the different types of software, we need to adopt different porting strategies.

### 2.2.2 Develop porting strategy

#### 2.2.2.1 Basic environment

operating system: version to support POWER architecture，i.e. Centos7.5 or above、RedHat7.5 or above. Please obtain the detail compatibility OS list from Inspur POWER System.  
（Contact Info：http://www.inspurpower.com/）

middleware: version to support POWER architecture, i.e. OpenJDK 8 or above

compiler: version to support POWER architecture, i.e. GCC 4.8.5 or above

#### 2.2.2.2 open source software/independent developed software

Each programming language contains a unique set of keywords and syntax, which are used to create a set of instructions. There are two types of High-level language. One type is Compiled language, the other type is Interpreted language. Compiled language requires the compiler to convert the high-level language instructions into machine code. Interpreted language requires translator to convert source code into the machine understood code.

|  |  |
| --- | --- |
| Compiling method | porting strategy |
| Compiled language | Requires source code to compile into machine code for POWER processor |
| Dependent Components：Obtain the source code for POWER and recompile it into components or replace it with similar components which support POWER architecture. |
| Interpreted language | Replace the interpreter for POWER architecture.  For example, JDK or Python parser |
| Dependent Components：Obtain the source code for POWER and recompile it into components or replace it with similar components which support POWER architecture. |

#### 2.2.2.3 Closed source software

For porting the closed source software, developers need to obtain the source code that supports POWER processor. If the source code is not available, please replace it with other similar software.

## 2.3 porting

Compare with porting software developed by compiled language, porting the software developed by interpreted language is relatively simple. We will focus on the porting process of software which developed by compiled languages, especially on compiling the build script and modifying the C/C++ source code.

### 2.3.1 Interpreted language

Translation by interpreter or parser: For those software which developed by interpreted language, you don’t need to modify any source code and recompile them. You just need to Interpret or parse the codes by interpreter or parser for POWER architecture. For example, OpenJDK for POWER or Python Parser for POWER.

Compile dependent library: If software contains compiled dependent libraries, please obtain the compiled dependent libraries for POWER architecture. If you can’t get them, please download the source code of related dependent libraries and recompile them on POWER platform.

### 2.3.2 Compiled language

#### 2.3.2.1 Prepare the compiling environment

Install compiler which support POWER architecture. Currently, the following compilers are available for POWER architecture:

GCC4.8.5 or above、 LLVM8.0 or above、 IBM XLC/C++、 IBM XLF、 IBM AT(IBM Advance-toolchain)

#### 2.3.2.2 Porting the compiler options

1）Add ppc64le options in configure file. For example, add ppc64le options in config.guess file.

2）Replace char variable with signed char variable. Add -fsigned-char option in Makefile.（the default char variable in x86 architecture is signed. The default char variable in POWER architecture in unsigned）；

3）POWER architecture does not support MMX/SSE/AVX instructions，please remove those compiler options, such as -msse、-msse4.1.

#### Porting compiler MACROS

1）gcc compiler defined MACROS for different architecture：replace \_\_x86\_64\_\_ or \_\_x86\_64 with \_\_powerpc64\_\_

2）User defined MACROS for different architecture： i.e. replace HAVE\_X86\_64 with HAVE\_POWERPC64

#### 2.3.2.4 Porting the source code

1. Porting SIMD instruction

a） porting MMX/SSE code: Upgrade the compiler to GCC 8+，add compiler option -DNO\_WARN\_X86\_INTRINSICS or modify the source code directly, replace the instructions with VSX set .

b） porting AVX code: Replace the instructions with VSX set.

2）Porting Assemble instruction

The assemble language for POWER architecture is totally different with X86 architecture. All code with developed by assemble language need to be replaced with assemble language for POWER architecture.

## 2.4 Optimizing

### 2.4.1 Compiler optimization

To gain better performance and deep optimization for POWER process, please obtain the following two commercial compilers:

1）IBM XLC/C++、IBM XLF compilers comes through in its optimization and the ability to improve code generation.

2）IBM AT(IBM Advance-toolchain) provides toolchain functionality earlier and a group of optimized libraries. AT is highly recommended when you want to build an optimized CPU-bound application on POWER or want some of the new toolchain functionalities on POWER before they make it into a distribution.

### 2.4.2 Compiler options optimization

GCC/AT compile option：-O3 -flto -funroll-loops -ffast-math -mcpu=power9 -mtune=power9

XLC/C++ compile option：-O3 -qhot -qipa -qpdf2 -qarch=pwr9 -qtune=pwr9

|  |  |  |
| --- | --- | --- |
| XLC/C++ compile options | GCC/AT compile options | Optimization |
| -O3 | -O3 | Better loop scheduling and transformation  Elimination of implicit memory usage |
| -qhot | -funroll-loops | Enables a set of high-order transformation optimizations that are most effective when optimizing loop constructs. |
| -qipa | -flto | IPA restructure application, performing optimizations such as inlining between compilation units and improve inline code |
| -qpdf1 -qpdf2 | -fprofile-generate -fprofile-use | Use PDF to get information such as the locations of heavily used or infrequently used blocks of code. |
| - | -ffast-math | Control compiler behavior regarding floating point arithmetic and allow optimizations for floating-point arithmetic. |
| -qarch=pwr9 | -mcpu=power9 | The most important step in influencing chip-level optimization, especially for POWER9. |

### 2.4.3 Library optimization

Please use components provided by IBM because IBM experts have optimized those components for POWER architecture. For example, IBM ESSL、IBM Spectrum MPI.

### 2.4.4 OpenJDK optimization

Compare with OpenJDK for X86 architecture, OpenJDK for POWER architecture added some options to optimize garbage collection：  
-XX:+UseParallelGC -XX:+UseAdaptiveSizePolicy -XX:ParallelGCThreads=4。

Compared with x86 platform, POWER platform need to add optimization option for garbage collection：-XX:+UseParallelGC -XX:+UseAdaptiveSizePolicy -XX:ParallelGCThreads=4。

### 2.4.5 Code optimization

1）Adjust alignment cache-line: To improve the efficiency of read/write cache and to prevent the impacts of performance from cache-line share issue. Developers can use cache-line adjustment in software code. The cache-line size is 128 Bytes in POWER architecture，but the cache-line size is 64 Bytes in X86 architecture. Those codes which use cache-line feature in POWER architecture need to adjust the size accordingly. For example:

struct {

int count \_\_attribute\_\_((aligned(128)));

} counts[N\_CPUS];

2）Adjust Alignment Page: To improve program performance, applications use data isolation based on memory pages to avoid memory resource competition issue during high concurrency environment. The page size is 64KB in POWER architecture, but the page size is 4KB in X86 architecture. The page size is set in some applications. In this situation, please modify those code and adjust the page size for POWER architecture. For example:

long page\_size =sysconf(\_SC\_PAGE\_SIZE);

void \*data;

size\_t data\_size = (size\_of\_data);

posix\_memalign(&data,page\_size, data\_size);