# 一、软件迁移的必要性

## 1.1 程序执行过程

所有的软件均由编程语言实现，编程语言在计算机上最终会被翻译成机器语言，以指令的形式在处理器上执行。

## 1.2 POWER处理器和x86处理器的指令差异

处理器能够识别的所有指令的集合就是指令集，而不同架构处理器所使用的指令集是不同的。POWER架构处理器使用的是基于RISC（Reduced Instruction Set Computing）的ISA指令集，而x86架构使用的是CISC（Complex Instruction Set Computing）指令集。

因此，x86架构下软件在POWER平台上运行必须经过相应的迁移。

## 1.3迁移过程中的常见问题类型

1. Makefile 文件及configure文件兼容性问题

Makefile文件中不同架构使用的编译选项可能会有所不用，如果makefile文件使用不当会导致编译问题，通常编译时会使用configure来生成makefile文件，因此需要在configure文件中加入相应的架构选项，在POWER平台编译时需要加入ppc64le架构选项

例如python2.7的configure.guess，需要在configure.guess中增加对POWER架构的支持

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. 汇编指令兼容性问题

不同平台使用的汇编指令是完全不同的，跨平台执行汇编指令会触发指令异常，因此在POWER平台编译软件时需要将软件代码中的X86平台的汇编指令执行进行相应的替换

如CPU序列号获取功能，需要对X86的汇编指令进行如下替换：

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：

从"/proc/device-tree/vpd/"系统文件读取

1. SIMD扩展指令兼容性问题

目前不同平台均可以通过SIMD（单指令多数据流）扩展指令集提供向量技术，但不同架构平台提供的指令集完全不同，X86平台使用MMX/SSE/AVX指令集,而目前POWER平台目前使用VSX指令集

如：popcnt指令，需要在代码中将X86的SSE指令集替换为PPC64le的指令

X86：

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

PPC64LE:

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

1. 内存页大小兼容性问题

不同架构平台使用的内存页大小有所差异，X86平台使用页大小为4k，POWER平台默认使用的页大小为64k，如果应用程序中使用了错误的页大小会导致不可预测的问题。

如mozjs17（Mozilla JavaScript 引擎）代码中将pagesize固化为4k导致在64k内存页大小的平台中运行时产生段错误，因此需要修改pagesize为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;

# 二、软件迁移步骤

整个软件迁移需要经过四个步骤：迁移准备、软件分析、编译迁移、优化。

## 2.1 迁移准备

POWER平台环境准备，可采用下列任意一种方式：

1. 向浪潮商用远程测试中心申请迁移测试环境；（联系方式：http://www.inspurpower.com/ziyuan/）
2. 本地可连接网络的POWER服务器。

## 2.2 软件分析

本阶段主要进行被迁移软件的软件栈分析及迁移策略制定。

### 2.2.1 软件栈分析

#### 2.2.1.1 基础环境分析

分析当前软件的基础环境，主要包括操作系统、中间件、编译器等，以便于后续软件迁移时尽可能的让POWER平台的基础环境与之对齐，尽量减少两者的差异。

#### 2.2.1.2 上层应用分析

分析软件包及其依赖组件，根据能否获得源码，将待移植软件及其组件分为开源/自研以及闭源两种类型，不同类型的软件采取不同的迁移策略。

### 2.2.2 迁移策略制定

#### 2.2.2.1 基础环境

操作系统：支持POWER架构的操作系统版本，如Centos7.5及以上、RedHat7.5及以上版本的操作系统，具体参见浪潮商用提供的OS兼容性列表（联系方式：http://www.inspurpower.com/）

中间件：支持POWER架构的相应版本，如OpenJDK 8及以上

编译器：支持POWER架构的相应版本，如GCC 4.8.5及以上

#### 2.2.2.2 开源/自研软件

对于开源/自研软件而言，可以根据实现软件编程语言类型的不同制定不同的迁移策略。高级编程语言通常可以分为两类：一类是编译型语言，一类是解释型语言。编译型语言通过编译器生成机器语言程序，机器语言程序可以直接被CPU执行；解释型语言通过对应的语言虚拟机或者解释器执行，语言虚拟机或者解释器屏蔽了不同CPU架构的差异。

|  |  |
| --- | --- |
| 翻译方式 | 迁移策略 |
| 编译型语言 | 源码：重新编译 |
| 依赖组件：获取POWER版本或者下载源码重新编译，闭源组件需要替换POWER版本或替换类似组件库 |
| 解释型语言 | 替换POWER版本的语言虚拟机或者解释器，如JDK或Python解释器 |
| 依赖组件：获取POWER版本或者下载源码重新编译，闭源软件需要替换POWER版本或者替换类似组件库 |

#### 2.2.2.3 闭源软件

对于闭源软件的迁移，需要获取支持POWER处理器的软件版本，如果无法获取，则替换其他类似软件。

## 2.3 编译迁移

由于解释型语言迁移相对简单，我们将重点介绍编译型语言的迁移过程，主要需要进行编译构建脚本和C/C++源码的修改。

### 2.3.1 解释型语言

直接翻译：对于解释性语言实现的程序，代码无需修改，程序也不需要重新编译，直接替换对应POWER版本的语言虚拟机或者解释器即可，如系统自带的OpenJDK及Python解释器。

依赖库编译：如果软件包含由编译型语言实现的依赖库，则需要获取依赖库对应的POWER版本，如无法获取可用的POWER版本，则需下载依赖库的源码包重新编译依赖库。

### 2.3.2 编译型语言

#### 2.3.2.1 编译环境准备

安装支持POWER平台的编译器，目前可用的编译器如下：

GCC4.8.5及以上版本、LLVM8.0及以上、IBM XLC/C++、IBM XLF、IBM AT(IBM Advance-toolchain)

#### 2.3.2.2 编译选项移植

1）configure文件中增加适配ppc64le编译选项，例如使用支持ppc64le架构的config.guess

2）显式定义char类型变量为有符号类型，在Makefile文件中添加-fsigned-char编译选项（x86默认是有符号数，POWER平台默认是无符号数）；

3）POWER平台不支持MMX/SSE/AVX指令，去掉-msse、-msse4.1等AVX编译选项。

#### 编译宏移植

1）gcc编译器自定义平台属性编译宏：将\_\_x86\_64\_\_或\_\_x86\_64替换为\_\_powerpc64\_\_

2）用户自定义平台属性编译宏：例如将HAVE\_X86\_64替换为HAVE\_POWERPC64

#### 2.3.2.4 源码移植

1. SIMD扩展指令集移植：

a） MMX/SSE代码适配：升级到GCC 8+，添加编译选项-DNO\_WARN\_X86\_INTRINSICS或者直接修改代码，使用VSX指令进行替换；

b） AVX代码适配：使用VSX指令进行替换。

2）汇编指令移植

POWER的汇编语言与x86的完全不同，所有涉及到使用汇编语言的代码，都需要基于POWER进行移植。

## 2.4 优化

### 2.4.1 编译器优化

可以将GCC升级为以下两种编译器：

1）IBM XLC/C++、IBM XLF，IBM发布的针对C/C++及Fortran的编译器，针对POWER平台进行了深度优化，提高应用程序运行效率

2）IBM AT(IBM Advance-toolchain)，IBM发布的充分利用POWER硬件特性的编译工具集，包含了最新版本的编译器（基于GCC）、更完整的功能库、DEUBG分析工具等

### 2.4.2 编译选项优化

GCC/AT编译选项：-O3 -flto -funroll-loops -ffast-math -mcpu=power9 -mtune=power9

XLC/C++编译选项：-O3 -qhot -qipa -qpdf2 -qarch=pwr9 -qtune=pwr9

|  |  |  |
| --- | --- | --- |
| XLC/C++编译选项 | GCC/AT编译选项 | 优化内容 |
| -O3 | -O3 | 提高代码的并行执行度，充分利用CPU的流水线、Cache等 |
| -qhot | -funroll-loops | 循环优化以及函数内联，可以得到较好的程序性能 |
| -qipa | -flto | 跨编译单元优化，适用于调用小函数特别多的场景 |
| -qpdf1 -qpdf2 | -fprofile-generate -fprofile-use | 反馈式编译优化，适用于大量的跳转或分支以及间接函数调用等 |
| - | -ffast-math | 触发non-standards-compliant浮点优化，适用于浮点精度要求不高的浮点计算优化 |
| -qarch=pwr9 | -mcpu=power9 | 针对POWER9的架构优化 |

### 2.4.3 组件库优化

使用IBM提供的针对POWER平台深度优化过的组件库，如IBMESSL、IBMSpectrum MPI。

### 2.4.4 OpenJDK优化

POWER平台与x86平台相比，需要增加针对垃圾回收的优化：-XX:+UseParallelGC -XX:+UseAdaptiveSizePolicy -XX:ParallelGCThreads=4。

### 2.4.5 代码优化

1）基于cacheline数据对齐调整：为提升cache读写效率，同时避免cacheline伪共享问题影响程序性能，可在程序代码中进行cacheline对齐。POWER上cacheline size是128 Bytes，x86 cacheline size是64 Bytes，代码需要针对POWER平台cacheline size大小进行相应调整。例如：

struct {

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

} counts[N\_CPUS];

2）基于内存页数据对齐调整 ：应用程序通过基于内存页的数据隔离以避免高并发时的内存资源竞争问题，提升程序性能。POWER上pagesize是64KB，x86 pagesize是4KB，一些应用中显式指定了pagesize的大小来实现，这种情况下代码需要针对POWER平台pagesize大小进行调整。例如

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

void \*data;

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

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