# 面向 C4350AL 处理器的 GCC 移植与优化①

陈 欣, 吴 伟, 陶秋铭, 赵 琛

(中国科学院 软件研究所, 北京 100190)

**摘 要**: 在分析 GCC 结构的基础上, 阐述了 GCC 在 C4350AL 处理器上的移植与优化方案. 针对 C4350AL 对 GCC 的 x86 后端进行了扩展, 实现了 GCC 对 C4350AL 的识别. 根据 C4350AL 的结构特性, 在 GCC 中为建立了流水线模型描述, 并基于 SPEC2006 测试程序对模型效果进行了验证. 实验表明采用该模型使得 GCC 在 C4350AL 上获得了性能的提升.

**关键词**: GCC; C4350AL; 后端移植; 优化

## Porting and Optimization of GCC on C4350AL

CHEN Xin, WU Wei, TAO Qiu-Ming, ZHAO Chen

(Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

**Abstract**: Based on the analysis of GCC's structure, the porting and optimization of GCC on C4350AL is presented. GCC's x86 backend is extended for the compiler's recognition of C4350AL. According to the properties of C4350AL's architecture, a processor pipeline model description is built. The model's effect is tested on SPEC2006 benchmark. The experiment results show that this model has improved GCC's performance on C4350AL.

**Key words**: GCC; C4350AL; backend porting; optimization

GCC(GNU Compiler Collection)[1] 是一套由 GNU 工程开发的支持多种编程语言的编译器. 它是自由软件发展过程中的著名例子, 由自由软件基金会以 GPL 协议发布, 是大多数类 Unix 操作系统(如 Linux、BSD、Mac OS X 等)的标准编译器. 它同样适用于微软的 Windows, 并且支持多种计算机体系芯片, 如 x86、ARM 等, 而且已移植到其他多种硬件平台. GCC 本只能处理 C 语言, 但其得到快速扩展, 支持处理 C++, 后来又扩展为能够支持 Fortran、Pascal、Objective-C、Java、Ada、Go 等.

C4350AL 是一款现代的超标量通用 CPU. 其设计理念是在在保证主流应用的体验下尽可能地实现低成本和低能耗. 在兼容 x86 指令集的基础上, C4350AL 还支持 SSE4 多媒体指令集、原生 64 位计算与 VT 虚拟化技术, 并提供了独有的 Padlock 硬件加密技术[2].

在进行针对性的移植之前, C4350AL 处理器对于 GCC 是一种 x86 家族的通用模型. 由于一个系统的性能, 不单单指 CPU 芯片的性能, 而是依赖于整个硬件设备、操作系统和编译器三方面的紧密耦合. GCC 虽然可以直接运行在 C4350AL 上, 但并没有针对其硬件结构做专门的优化, 因此其性能不能得到完全发挥, 所以需要对 GCC 进行 C4350AL 的针对性移植和优化.

本文分析了 GCC 编译系统的结构以及其移植机制, 并对 C4350AL 的体系结构进行了介绍, 讨论了针对 C4350AL 的 GCC 移植和优化过程, 并基于 SPEC2006 基准测试程序分析了优化前后的 GCC 性能, 证明了优化工作的效果.

## 1 GCC结构分析

如图 1 所示, GCC 编译器由三部分组成: 语言前端, 与目标平台无关的中间代码优化器以及与平台相关的

① 基金项目:国家自然科学基金(61100067);中国科学院战略性先导专项课题(XDA06010600)

收稿时间:2015-01-08;收到修改稿时间:2015-03-25

中间代码生成器. 这种前端与后端分离的设计非常适合对多种不同的程序语言和目标平台进行扩展和支持. 因此 GCC 编译器具有极强的可移植性和可扩展性.

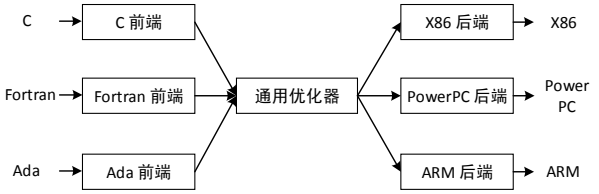


图 1　GCC 编译器基本架构

在 GCC 编译器的基本设计架构中, 中间代码的生成和优化是一个核心设计. 中间代码是一种与高级语言程序等价, 而与目标平台无关的代码. 通过这样的设计, 无论 GCC 需要编译何种语言, 在通过前端处理之后都会转换成中间代码. 因此只要针对某一个目标平台进行移植后, 该平台就可以支持多种高级语言的编译.

### 1.1 后端的实现

GCC 对于后端的描述分为三个主要文件, 一个 C 语言宏定义头文件 machine.h, 一个 C 源文件 machine.c 以及一个用 RTL 表达式写成的机器描述文件 machine.md, 这三个文件定义了机器的指令集、功能部件以及延迟特性等[3].

对于 x86 族处理器, GCC 定义了一个公用的宏文件 i386.h, 其中包含了对各种处理器(Intel、AMD 等)的参数描述: 编译器运行环境、基本机器属性、ABI 及机器描述的支持等. 在 i386.md 中则给出了通用的指令定义、指令属性的定义、指令拆分的定义以及指令优化的定义等. 在 i386.c 中则包含一些与宏同名的函数扩展的 c 程序以及机器描述中特殊功能的定义. 此外, GCC 在 driver-i386.c 文件中定义了一系列方法, 用于在编译的过程中, 根据处理器的制造商信息、序列号、家族、模型系列等信息来确定 CPU 的型号[4].

## 2　C4350AL体系结构分析

C4350AL 是具有 7 个执行单元的超标量处理器, 如图 2 所示, C4350AL 主要由 5 大模块组成:

顺序读取翻译模块: 这个流水线模块负责顺序读取 x86 指令并将其解码成内部的机器指令, 称为微指令(micro-ops), 然后送到下一个模块. 每个时钟可以解码 3 条全 x86 指令.

指令乱序分配执行模块: 该流水线模块负责将翻译后的微指令分配给合适的执行单元. 这个分配和执行的过程并不一定遵循程序的顺序完成, 只要一个指令的前置输入可用了, 它就会被分配和执行(因此称之为乱序). 乱序分配执行的实现得益于重排缓存(Reorder Buffer, ROB)和内存重排缓存(Memory Reorder Buffer, MOB)

顺序撤出模块: 重排缓存中已经完成执行的微指令会按程序顺序撤出. 微指令撤出意味着指令执行生效引起的各种改变: 寄存器和内存得到更新, 异常得到处理等等.

缓存子系统: 缓存子系统由 1 级指令缓存(I-cache), 1 级数据缓存(D-cache), 2 级缓存(L2-cache)以及一些其它的特殊缓存组成.

电源及热管理模块: 电源管理组件分布在整个 CPU 的所有部件中. 某些部件(例如时钟门、总线管理等)可以减少处理器逻辑消耗的能源.

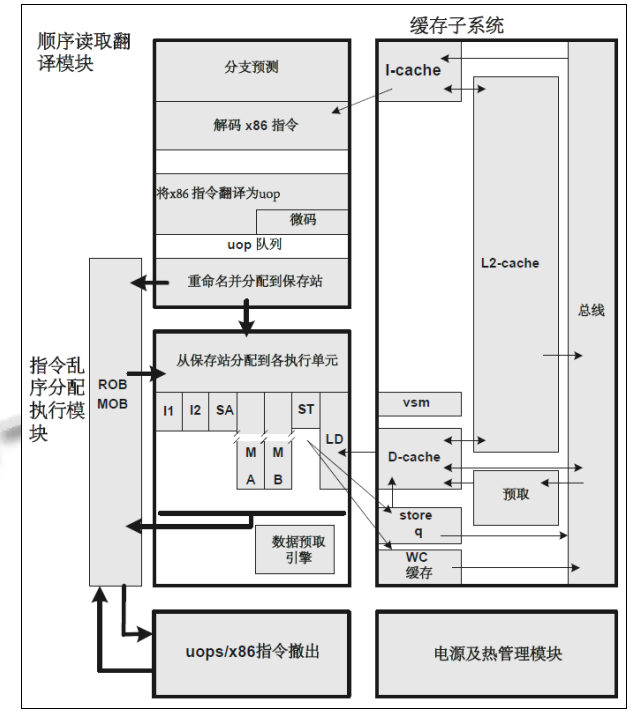


图 2　C4350AL 结构图

指令执行端口(execution port)

在针对 GCC 的移植过程中, C4350AL 的指令分配执行模块尤为关键, 其 7 个执行端口(对应图 1 中的 I1、I2、SA、MA、MB、ST、LD) 分别负责不同的功能, 后文在 GCC 中为 C4350AL 建立处理器流水线描述的过

程中, 会根据这些信息为各个端口建立模型, 用于刻画不同指令对处理器资源群的占用情况. 执行端口列表如表 1.

表 1 C4350AL 执行端口列表

| 端口 | 功能 |
|---|---|
| I1 端口 (ALU 1) | 执行所有非 SIMD 整数指令(除了乘除): 加法/减法、逻辑、偏移、移动、位扫描等等. |
| I2 端口 (ALU 2) | 与 ALU1 端口相同. |
| SA 端口 (Store Address) | 执行存址运算以及 LEA 指令. |
| SD 端口 (Store Data) | 执行数据存储操作. |
| MA 端口 (Media-A) | 该端口下连接了一系列子端口, 分别执行浮点加法、普通浮点操作(例如移动、浮点逻辑运算等)、除法和平方根运算以及独有的数据安全操作等. |
| MB 端口 (Media-B) | 该端口主要设计用于执行高性能乘法, 包括整数和浮点运算. 高性能乘法包括单精度 SSE 浮点乘法、双精度 SSE 浮点乘法以及 x87 浮点乘法等. |
| LD 端口(Load) | 执行装载操作. |

## 3 GCC移植与优化实现

由于 x86 家族指令相关的通用定义都已经在公用的机器描述中被定义过, 无需再重复定义. 为了完成针对 C4350AL 的 GCC 移植, 我们只需要在 x86 家族处理器的描述中为 C4350AL 添加 CPU 入口, 即根据 C4350AL 特有的硬件参数实现 GCC 对 C4350AL 的识别. 在此之后, 我们根据 C4350AL 的结构为其编写了处理器流水线描述, 使得 GCC 在编译过程中可以针对 C4350AL 的结构特点来进行指令调度, 达到编译性能优化的目的.

### 3.1 添加后端入口–CPU 的识别

移植的第一步是在 x86 后端系列中为 C4350AL 添加定义, 然后通过 CPUID 汇编指令, 我们可以获取 C4350AL 的类型, 型号, 制造商信息, 商标信息, 序列号, 缓存等一系列 CPU 相关信息. 之后将这些信息分别填写到对应的后端文件中, 即可实现 GCC 对 C4350AL 的自动识别. x86 后端的文件路径为 /gcc/gcc/config/i386/, 具体添加内容包括:

①在 i386.h 文件中添加宏定义和处理器类型定义

添加 C4350AL 宏定义:

```
#define TARGET_C4350AL (ix86_tune == PROCESSOR_C4350AL)
```

添加处理器类型定义:

在枚举类型 enum processor_type 中添加处理器类型:

```
enum processor_type
{
  PROCESSOR_GENERIC = 0,
  ……
  PROCESSOR_C4350AL,
  PROCESSOR_max
};
```

②在 i386.c 中添加处理器模型定义信息与架构定义

添加特性/优化位掩码:

```
#define m_C4350AL (1<<PROCESSOR_C4350AL)
```

在枚举类型 enum processor_model 中添加处理器模型:

```
enum processor_model
  {
    M_INTEL = 1,
    M_AMD,
……
    M_C4350AL
  };
```

在数组 arch_names_table 中添加处理器架构名:

```
const arch_names_table[] =
    {
      {"amd", M_AMD},
      {"intel", M_INTEL},
      ……
      {"C4350AL", M_C4350AL},
    };
```

③在 driver-i386.c 中添加针对 C4350AL 的判断信息

根据 CPUID 指令的结果, CPU family 值为 6, model 值为 15, 我们在 driver-i386.c 中添加对应的判断语句如下:

```
switch (family)
{
  case 6:
```

```
    if (model > 9)
    {
     if (model == 0xf)
          cpu = "C4350AL";
       else
          processor = PROCESSOR_GENERIC;
}
```

### 3.2 优化 GCC 的指令调度

#### 3.2.1 处理器流水线模型原理

为了获得更好的性能, 大多数现代处理器都有许多功能单元用于同时处理多条指令. 在处理器中, 由于指令对功能部件的依赖会引起流水线的延迟, 包括各种共享的处理器资源, 例如内部寄存器、总线等等, 指令在条件满足的情况下会得到执行, 否则它就会被暂停直到其执行条件得到满足, 这种延迟较为复杂.

对于不同的目标处理器, GCC 提供了流水线模型[5]来描述它们的并行处理能力以及不同指令对处理器资源群的占用情况[6], 从而较精确的刻画数据相关和资源相关的情况. GCC 会根据流水线描述文件自动生成流水线相关识别器[7](pipeline hazard recognizer), 其本质是一个有限状态自动机(deterministic finite state automation: DFA). 该 DFA 的作用是描述目标机器中的资源使用状况, 当存在一个状态转换时, 表明某条指令需要的资源就绪, 相应的指令可以发射. 在进行指令调度时, GCC 使用 DFA 来找出在给定的始终周期可以发射的指令, 来更新就绪队列.

GCC 提供的主要流水线描述规则如图 3 所示:

```
规则 1: define_automaton automata-names (定义自动机)
规则 2: define_cpu_unit unit-names [automaton-name] (定义功能部件 unit-names 属于自动机)
规则 3: define_insn_reservation insn-name default_latency condition regexp (定义指令 insn-name 的延迟 default-latency 以及通常情况下占用的资源群 reg-exp)
规则 4: exclusion_set unit-names unit-names (两个功能部件不能同时使用)
```

图 3　GCC 流水线模型描述规则

#### 3.2.2 添加 C4350AL 的处理器流水线模型

为 C4350AL 建立流水线模型, 可以使 GCC 在指令调度时, 考虑到各功能部件的冲突, 从而对指令进行更合理的调度, 生成效率更高的目标代码. 因此我们根据 GCC 提供的流水线定义规则以及 C4350AL 的体系结构特征, 在 X86 后端系列中为其添加了针对 C4350A 的机器描述文件 c4350al.md, 首先通过规则 1 定义了六个自动机, 如图 4 所示, 接着是关于各类指令实际占用资源群的定义, 其中一条示例如图 5 所示.

```
(define_cpu_unit "n2_p0,n2_p1 " "C4350AL_core")
说明: 对应 I1、I2 端口
(define_cpu_unit "n2_p2" "C4350AL_load")
说明: 对应 LD 端口
(define_cpu_unit "n2_p3,n2_p4" "C4350AL_store")
说明: 对应 SA 和 SD 端口
(define_cpu_unit "n2_idiv" "C4350AL_idiv")
说明: 为整数除法器单独设置一个自动机
(define_cpu_unit "n2_fdiv" "C4350AL_fdiv")
说明: 为浮点除法器单独设置一个自动机
(define_cpu_unit "n2_ssediv" "C4350AL_ssediv")
说明: 为 SSE 除法器单独设置一个自动机
```

图 4　流水线描述中功能部件的定义

```
(define_insn_reservation "n2_push_reg" 1
      (and (eq_attr "cpu" "C4350AL")
           (and (eq_attr "memory" "store")
           (eq_attr "type" "push")))
           "n2_decodern,n2_p4+n2_p3")
说明: 表示 push 到寄存器操作在第一个周期会占用一个解码部件, 在第二个周期会占用 SA 和 SD 端口. 该操作的延迟时间为 2 个周期.
```

图 5　流水线描述中指令的定义

## 4　移植与优化结果验证

本节基于测试程序集 SPEC2006(Standard Performance Evaluation Corporation 2006)[8]来验证了 CPU 流水线模型的效果. 实验环境如表 2 所示.

表 2　实验环境

| | |
|---|---|
| 处理器 | C4350AL @ 1.6+ GHz |
| 内存 | 4G DDR3 |
| L1 D-Cache | 64KB 16-路组相联 |
| L2 Cache | 1MB 32 路组相联(互斥) |
| FSB | 1333MHz |
| OS | Ubuntu14.04 32bit |
| 编译器 | GCC 4.8.4 |

### 4.1 SPEC2006 测试集介绍

SPEC 全称为标准性能评估组织, 是由计算机厂商、系统集成商、大学、研究机构、咨询等多家公司组成的非营利性组织, 这个组织的目标是建立、维护一套用于评估计算机系统的标准. 其中 SPEC CPU 2006 是 SPEC 推出的 CPU 子系统评估软件, 旨在提供 CPU、内存和编译器综合的整体性能度量, 用来比较不同计算机系统密集型计算的工作负载. SPEC CPU 2006 包含两

个基准套件:测量整数计算密集型性能的 CINT2006 和比较测量浮点计算密集型性能的 CFP2006.

### 4.2 优化前后性能测试与分析

在 GCC 中为 C4350AL 添加了入口后, 我们可以使用选项-mtune=C4350AL 来主动开启针对 C4350AL 的优化. 使用 GCC 编译器在 C4350AL 上编译、执行标准测试程序 SPEC2006 后, CINT2006 的结果如表 3 所示, CFP2006 的结果如表 4 所示:

表 3 优化前后 CINT2006 结果对比

| CINT 2006 | 优化前分值 | 优化后分值 | 性能变化(%) |
|---|---|---|---|
| 400.perlbench | 9.98 | 9.99 | +0.1 |
| 401.bzip2 | 5.47 | 5.49 | +0.4 |
| 403.gcc | 8.82 | 8.84 | +0.2 |
| 429.mcf | 8.27 | 8.27 | 0 |
| 445.gobmk | 9.48 | 9.54 | +0.6 |
| 456.hmmer | 6.8 | 6.8 | 0 |
| 458.sjeng | 9.44 | 9.47 | +0.3 |
| 462.libquantum | 13.2 | 13.2 | 0 |
| 464.h264ref | 11.1 | 11.1 | 0 |
| 471.omnetpp | 7.22 | 7.32 | +1.4 |
| 473.astar | 5.54 | 5.56 | +0.4 |
| 483.xalancbm | 9.14 | 9.14 | 0 |
| 平均 | 8.45 | 8.47 | +0.1 |

从 CINT2006 的结果可以发现, 对 11 个整点程序, 9 个程序的变化在 0.5%以内, 可以忽略, 剩下 2 个程序的性能上升超过了 0.5%, 总体性能提高了 0.1%

表 4 优化前后 CFP2006 结果对比

| CFP 2006 | 优化前分值 | 优化后分值 | 性能变化(%) |
|---|---|---|---|
| 410.bwaves | 8.26 | 8.32 | +0.7 |
| 416.gamess | 20.7 | 20.9 | +1.0 |
| 433.milc | 7.13 | 7.11 | -0.3 |
| 434.zeusmp | 3.08 | 3.08 | 0 |
| 435.gromacs | 2.48 | 2.49 | +0.4 |
| 436.cactusADM | 5.12 | 5.2 | +1.6 |
| 437.leslie3d | 5.56 | 5.56 | 0 |
| 444.namd | 5.9 | 5.93 | +0.5 |
| 447.dealII | 10.7 | 10.7 | 0 |
| 450.soplex | 7.21 | 7.06 | -2.1 |
| 453.povray | 8.72 | 8.73 | +0.1 |
| 454.calculix | 2.43 | 2.44 | +0.4 |
| 459.GemsFDTD | 5.21 | 5.26 | +1.0 |
| 465.tonto | 3.8 | 3.81 | +0.3 |
| 470.lbm | 10.1 | 10.1 | 0 |
| 481.wrf | 3.55 | 3.55 | 0 |
| 482.sphinx3 | 10.1 | 10.1 | 0 |
| 平均 | 5.58 | 5.59 | +0.2% |

从 CFP2006 的结果可以发现, 对 17 个浮点程序, 11 个程序的变化在 0.5%以内, 可以忽略, 5 个程序的性能上升超过了 0.5%, 而 1 个程序性能下降超过了 1%, 总体性能提高了 0.2%. 部分程序的性能有所下降, 这是因为我们无法精准的描述每一种指令的资源群占用状况.

总的来说, 应用 C4350AL 的流水线模型之后, SPEC2006 的部分程序性能获得了提升, 因此对程序的性能改善是有益的.

## 5 结语

为了最大化 C4350AL 处理器在 Linux 操作系统下的性能, 需要针对其进行 GCC 的移植和优化. 本文根据 C4350AL 的结构特性, 为其在 GCC 的 x86 后端中进行了扩展, 使得 GCC 可以识别 C4350AL, 从而实现了 GCC 在 C4350AL 上的移植. 此外, 为了优化 GCC 的指令调度, 我们为 C4350AL 建立了处理器流水线模型描述, 并通过 SPEC2006 测试程序 集, 证明应用该模型使得 GCC 编译后的整数计算密集型程序性能和浮点计算密集程序性能分别提高了%0.1 和 0.2%, 初步达到了优化目标.

## 参考文献

1 GCC 官方网站. http://gcc.gnu.org/projects/ia64.html.

2 Henry GG. The VIA Isaiah Architecture. Centaur Technology, Inc. 2008. 1–12

3 Vichare A, Deshpande S. GCC implementation. Indian Institute of Technology. Bombay. 2008.

4 吕鹏伟,袁成军,贺骊.Gcc编译器后端移植技术.现代电子技术,2012,35(6):39–42.

5 GNU Compiler Collection Internals(For gcc version 4.8.4). 2013. 403–409

6 Makarov VN. The finite state automaton based pipeline hazard recognizer and instruction scheduler in GCC. Proc. of the GCC Developers Summit. 2003. 423–432.

7 叶崴,马杰,侯朝焕.Gcc的流水冲突识别器和并行调度器.计算机工程与应用,2005,41(20):10–11.

8 SPEC2006 官方网站. http://www.spec.org/cpu2006/.