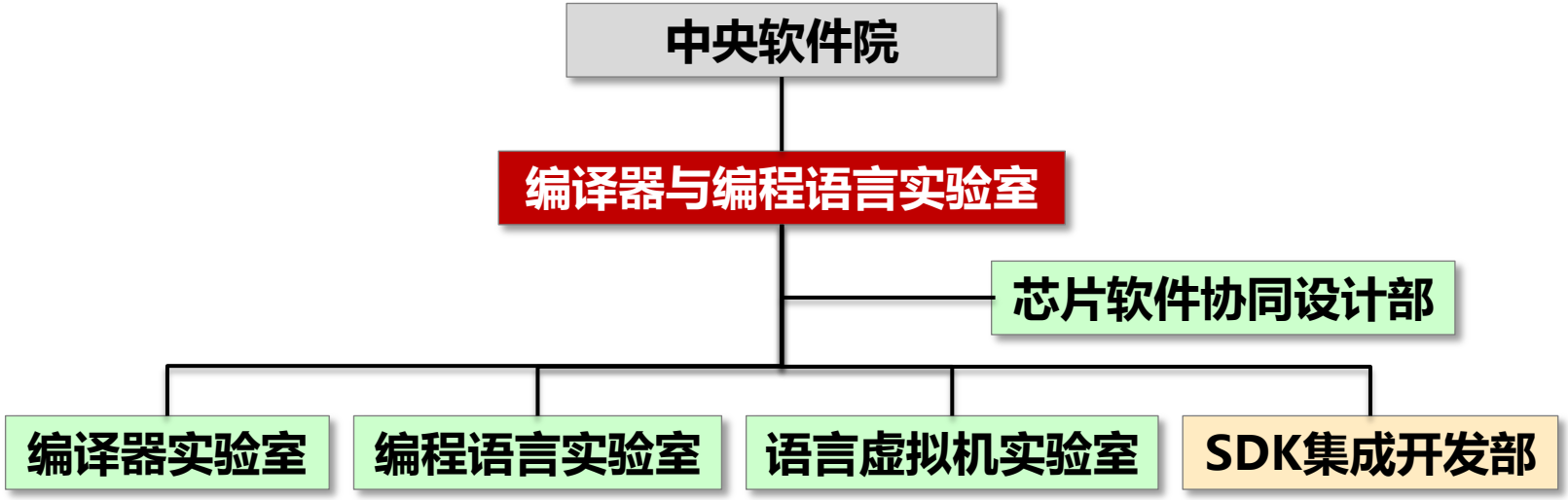
华为自研编译器探索与实践

华为公司编译器与编程语言实验室

www.huawei.com

华为编译器与编程语言LAB创新组织结构

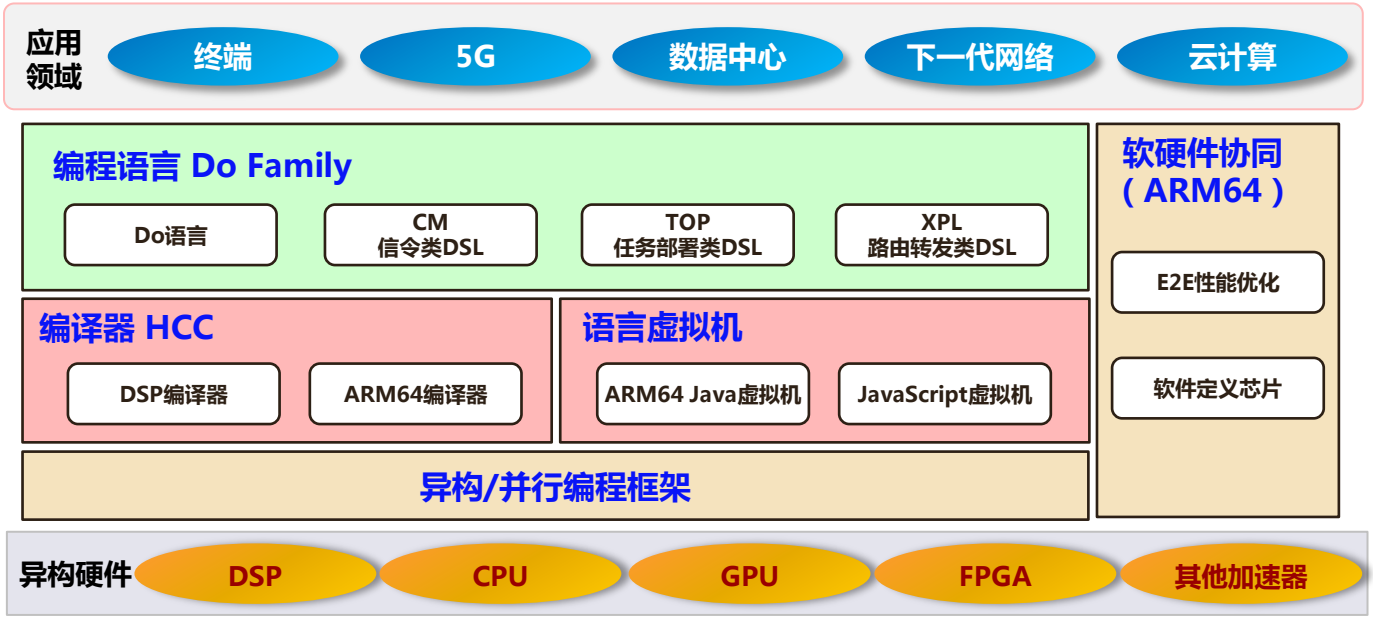


- 2016年8月正式开工运作（前身编译器团队）

技术创新

商用交付

华为编译器与编程语言业务



编译器：与芯片垂直优化，发挥自研芯片多核优势，形成差异化竞争力；
编程语言：面向业务编程，支撑业务快速创新，解决软件产能/效率/安全问题；
语言虚拟机：面向多语言、多硬件，提供性能/体积领先的虚拟机框架；
软硬件协同：深度软硬件协同，促进ARM64系统竞争力提升；

华为编译器发展路标(2009-)

第一代：
完成构建自研DSP、商用CPU编译器差异化技术

CPU：
- ARM32场景性能领先商用ARMCC
- X86场景性能领先商用ICC
- 提升无线控制器规格15～20%，超越友商

DSP：
- 性能超越标杆XCC，基站竞争力超越友商
- 单芯片节约定制费670万美元

第二代：
完成构建自有IP核DSP、自研ARM32编译器技术

CPU：
- 解决自研ARM编译器有无的问题，突破ARM32 PGO技术、指令跳转技术，自研编译器+自研ARM32竞争力领先
- 加速IPL32技术，存量代码“零代价”迁移到ARM64

DSP：
- 真正具备研发自有IP核DSP(包含核设计和编译器)的能力

第三代：
构建ARM64 C/C++编译器技术、领域定制DSP编译器技术

CPU：
- 自研ARM C/C++编译器实现ARM64领域竞争力最优(PMU, FGO, 定制指令)

DSP：
- DSP发展趋势为功能可定制、软件化，构建领域定制的DSP核心编译器和芯片软硬件协同技术，完成未来SoC、CPU+加速器领域的编译技术储备

2009，从0构建 | **2011，规模商用** | **2013，持续领先** | **2015，战略基石**

未来：
- 静态编译器技术储备布局基本完整、重点为基于业务的软件定义芯片能力
- 编译器下一步的重点开始围绕 **ARM64、众核、异构**

阶段一（2016）
➢全面构建DSP/CPU编译器核心竞争力，保证DSP/CPU芯片竞争力ready
➢业务场景： 5G、终端、固网

解决DSP/CPU芯片匹配优化问题
编译器竞争力比齐业界

阶段二（2017～）
➢支撑新语言的静态编译器
➢构建异构多核的编译优化技术（CPU/DSP/GPU/FPGA）
➢业务场景：5G、终端、数据中心、云计算

众核异构编译优化及架构指导技术
静/动态编译优化能力追赶业界

自研虚拟机简介

基于OpenJDK构建自研JVM

ARM64 JVM

2014 – 开始技术构建
- 基于OpenJDK

2015 – 补齐可靠性/性能
- 企业级应用的可靠性；
- P660(1P 32core)与Intel E5-2650V3相比的Java性能从40%提升至72%；
- 大数据Hadoop性能持平Intel E5-2650V3；

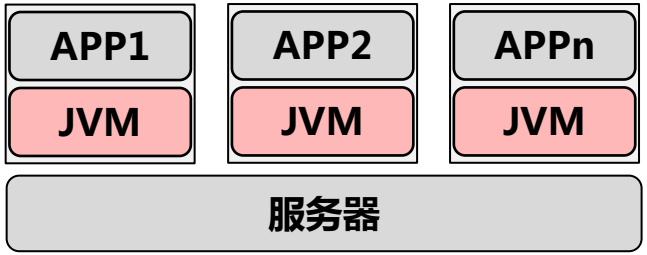
ONT JVM - 2015率先POC
- 体积和RAM占用只相当于Oracle JVM的95%；
- 每两年节省购买商用License费用XX万美元以上；

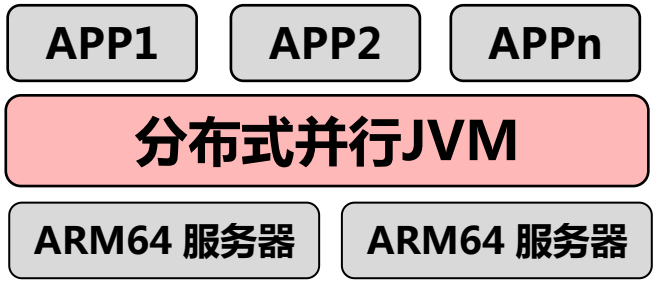
POC场景(**2016** -)
- 心声社区
- 公有云
- EMUI

2014 — 2015 — 2016

软件创新使能自研ARM64

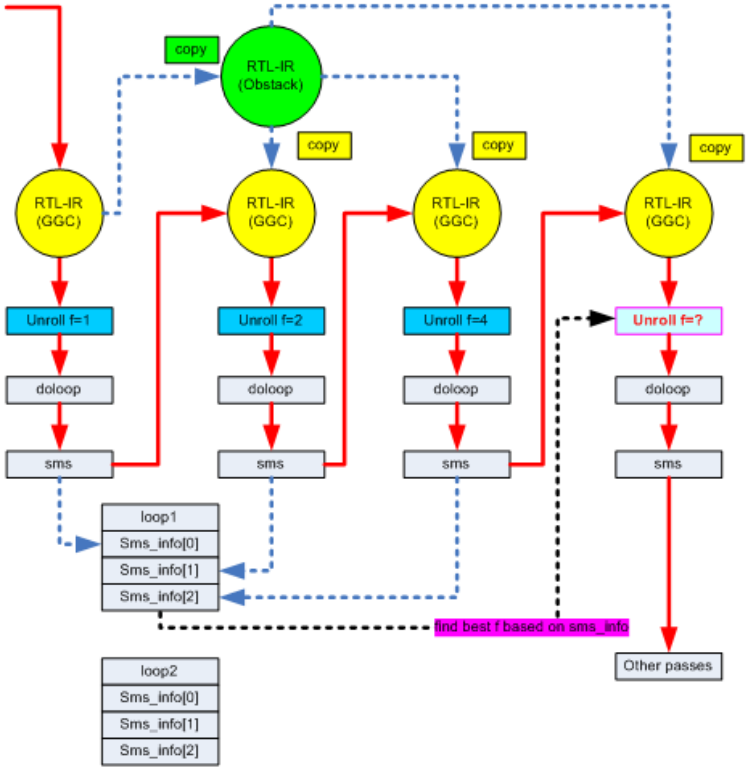
烟囱式架构，资源不能共享





- **资源全局共享**
- **性能线性扩展**
- **软硬协同优化**

自研编译器对开源编译器的架构扩展



- ✓ sms(软件流水) 需要多方面的能力：unroll，调度，alias，指令系统
- ✓ 开源编译器由社区大公司主导，通用面、广泛收益的贡献兴趣大。对极致精细的调度和控制缺乏兴趣和耐力。
- ✓ 如何在开源编译器下构筑自研能力：
  - 要动(目标清晰，分析准确)
  - 敢动(懂)
  - 能动(软件基本功)
  - 会动(敏捷软件基本开发方法)

From: Vladimir Makarov <vmakarov <at> redhat.com>
Subject: **Re: A GGC related question**
Newsgroups: **gmane.comp.gcc.patches**
Date: Monday 25th November 2013 03:17:26 UTC (over 3 years ago)

On 11/22/2013, 8:04 PM, dxq wrote:
> fixing SMS, do you mean that we only modify the SMS pass?
You don't need loop unrolling when you have a good software pipelining and loop vectorization. A good software pipelining can see through any number of iterations and has no problems with code cache locality in comparison with unrolling. A good SFP is a perfect SFP (e.g. resource constraints software piplening) which can deal with any loop bodies.

Unfortunately modulo scheduling deals with one BB loop body only (still it covers most cases after if-conversion, it is easy and can be good for supporting hardware with rotating regs file). Even if the current SMS implementation pitfalls are fixed, there are still cases when unrolling could be beneficial. So probably your approach is the best what can be done for now. Also if you manage to implement the infrastructure with copying/backup, it could be useful for a perfect SFP implementation. Still I think we need long-term strategy for SFP in GCC.

```
*presms
*all-presms
   rtl-into_cfglayout
   rtl-jump
   rtl-subreg1
   rtl-dfinit
   rtl-cse1
   rtl-fwprop1
   rtl-cprop1
   rtl-rtl pre
   rtl-hoist
   rtl-cprop2
   rtl-store_motion
   rtl-cse_local
   rtl-ce1
   rtl-reginfo
   rtl-loop2
      rtl-loop2_init
      rtl-loop2_invariant
      rtl-loop2_unswitch
      rtl-loop2_unroll
      rtl-loop2_doloop
      rtl-loop2_done
   rtl-web
   rtl-cprop3
   rtl-cse2
   rtl-dse1
   rtl-fwprop2
   rtl-auto_inc_dec
   rtl-init-regs
   rtl-ud_dce
   rtl-combine
   rtl-ce2
   rtl-bbpart
   rtl-regmove
   rtl-outof_cfglayout
   rtl-split1
   rtl-subreg2
   rtl-no-opt dfinit
   *stack_ptr_mod
   rtl-mode_sw
   rtl-asmcons
   rtl-sms
*pass_sms_unroll
```

自研编译器在芯片架构级软硬件协同应用

场景：静态变量load

动机：全局变量存取广泛存在场景，是否存在收益？

- 读取/写入一个数据的过程：
  - 读取变量的指针
  - 从指针处load/store值
- 读取指针是指令：l32r(const pool), movi32(long instruct world)或者已经存在的指针
- Load/store 是指令：l32i/s32i/l32x/s32x…

如果地址是常数，步骤可以简化。 比如x86的mov指令：

	Clocks			
operands	286	386	486	Size Bytes
reg,reg	2	2	1	2
mem,reg	3	2	1	2-4
reg,mem	5	4	1	2-4

```
movi32    a0, a
l32iz     a1, a0
movi32    a0, b
l32iz     a0, a0
addspi    -48
tstge     a0,a1
movif     a1, 0
movi32    a0, c
movit     a1, 1
s32i      a1, a0, 0
l32iz     a0, a0
addspi    48
ret
```

```
l32c a1, a
l32c a0, b
addspi -48
testge a0,a1
movif a1,0
movit a1,1
s32c a1,c
l32c a0,a1
addspi 48
ret
```

自研编译器在芯片架构级软硬件协同应用

- 基于trace的分析
  - 获得const pool的trace以及object文件，链接时的dump信息
  - 收集.o文件中的.literal信息段
  - 从链接文件中，获取.literal段里面每个 symbol的确定地址
  - 扫描trace里面的l32r a*, [symbol地址]
  - 扫描后续a的use指令
  - 统计结果
- 比较复杂的一个过程
- **Relax linking造成我们有很多的错误，关闭了relax linking得到正确结果**
- 反反复复和核架构师沟通，确保脚本分析和眼睛分析结果一致。

Subject: 答复: New instruction proposal for absolute/direct addressing, or ld/st global symbols directly

Hi, Alex:

As a whole scalar solution, how about add

l8ua, l16ua, l16sa, l32a, s8a, s16a, s32a

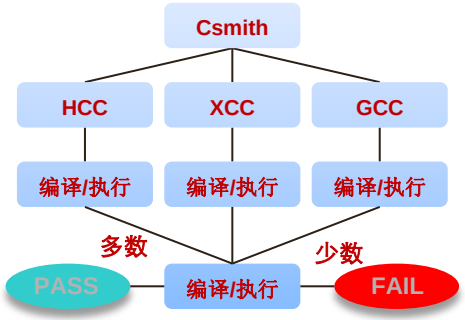
these 7 instructions?

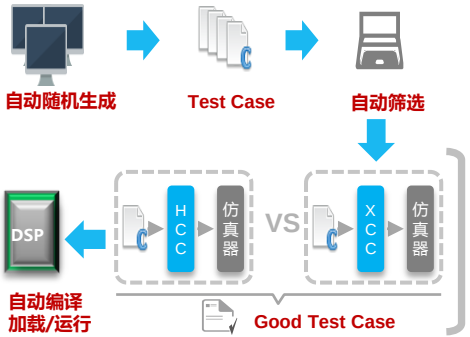
Even in the trace, we see at least 552+749+211+19+9 = 1540 useful cases.

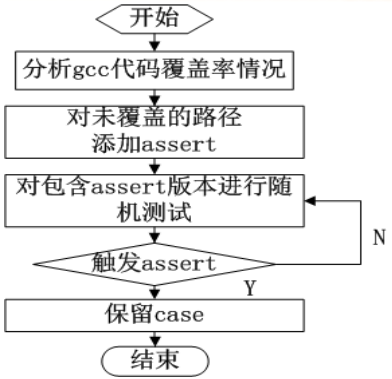
Thanks
Gang

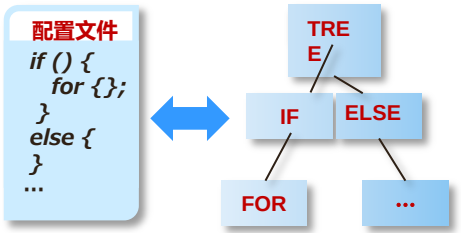
```
[yugang@dsp quantitive]$ tclsh quan.tcl
There are 714 real symbols(address) in BM_Sclar project production code.
  3858 address readings in trace for                                    CCH_SwiCP,     3234 readings may be eliminated.
     0 address readings in trace for                            LBB_Scalar_Jump_Fun,        0 readings may be eliminated.
    53 address readings in trace for                 CCH_To_DCH_Normal_2nd_DefadFactor,       33 readings may be eliminated.
    70 address readings in trace for                        CCH_Channel_Setup_Msg_Proc,       63 readings may be eliminated.
    12 address readings in trace for          LBB_UL_CTRL_ADMSCH_PushAgent1x2OMx8Proc,        0 readings may be eliminated.
    76 address readings in trace for                   HAC_3OSymbFHT_DecodeType_3OET,       47 readings may be eliminated.
     6 address readings in trace for                      UBB_ALG_CCH_FadeParaConfig,        6 readings may be eliminated.
    92 address readings in trace for                      LBB_PUSCH_DMRSCHE_ParaCalc,       52 readings may be eliminated.
    24 address readings in trace for                                    uniSemPend,       18 readings may be eliminated.
l8ui 552) (add 1301) (l32i 749) (s32i 211) (addx4 286) (addx8 36) (s8i 19) (addmi 117) (or 96) (addx2 59) (l16ui 9) (addi 11) (lv128.i 7)
```

自研编译器的自动化测试实践









自研编译器与产品联合优化提升产品价值

LTE面临友商激烈的竞争压力：
- 单板小区规格数提升
- 单TTI上下行调度用户数提升
- 小区支持连接数提升

基线版本局部负载接近饱和：
基于6182的新单板在eRAN7.0的交付规格上，经过多轮优化，局部负载仍接近饱和。

优化手段不够丰富：
单从业务层面优化无法满足芯片性能竞争力目标达成。

构建竞争力

跨部门联合运作

跨部门联合任命，明确职责、目标；以**项目迭代**方式运作，阶段目标达成对齐闭环；优化前培训赋能，优化后成果[illegible]鉴。

负载痛点挖掘

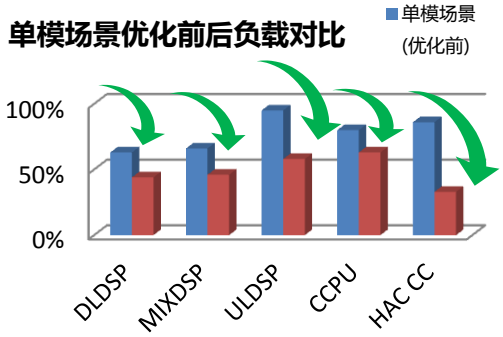
热点函数分析工具&板上打点工具精准定位热点函数及负载瓶颈；针对关键负载痛点，深入交流，明确优化策略。

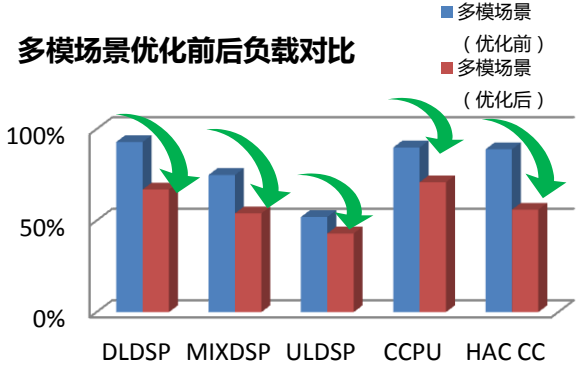
组织资产积累

开发**优化点提示工具**，固化优化方法，提升优化点发掘效率；总结优化过程中的**策略和方法**，在无线基带持续展播。

垂直拉通优化

优化基带算法，提升软件处理能力；应用**编译优化技术**，充分挖掘硬件能力；提出**芯片设计建议**，芯片能力持续演进。





编译器技术

优化点1： 合并大循环内若干小循环，减少对中间数据的存取。	单次循环减少5cycle，约提升5%	上库
优化点2： 使用HFMAX指令代替分支判断（已确定不会出现+/-0）； **优化点3：** 使用HFZERO_8X16代替HFREP(0)，减少对AR寄存器的使用和指令数目； **优化点4：** 使用SELI代替SEL，减少literal使用；	单次调用由99cycle减为81cycle，约提升22%。	上库

- 核平均负载优化XX%，为6182芯片基带业务演进提供性能储备，**降低成本，提升基带产品竞争力。**

自研编译器异构并行技术在热销产品中的应用

HDR算法优化技术方案：基于现有架构做性能调优，逐步使能异构协同框架，通过异构加速提升Camera整体性能

拍照处理流程：

- 前处理
- …
- 中处理
- …
- 后处理
  - HDR
  - deblur
  - …
- 图片分析
- …

HDR算法优化技术方案

- CPU
- GPU
- DSP

1、同构单核极致性能优化

(预期收益30%～50%)

- 指令预取
- 访存优化
- 循环展开（横向、纵向）
- 多线程并行
- 减少计算量
- 数据流并行

2、CPU+GPU异构核协同优化

(预期收益10%～20%)

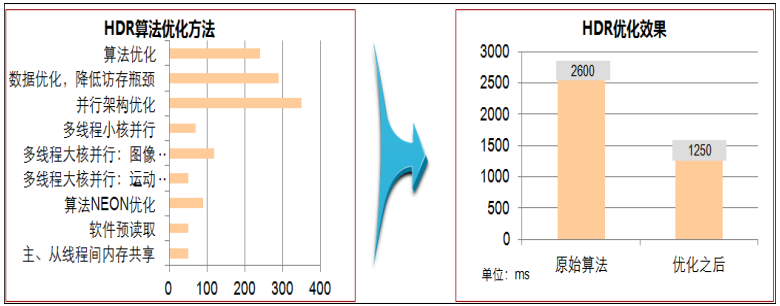
- 零拷贝
- OpenCL（CPU+GPU）
- 带宽限制
- CPU/GPU负载均衡

3、CPU+GPU+DSP协同优化

预期收益20%～30%

- OpenVX（DSP+GPU+CPU）
- 资源动态调度最优
- 最优的API适配

2016年面向终端的重点工作



2016年上半年成果：

1）deblur算法优化提升10+%，**落地P9产品**；JPEG图库解码性能提升10+%，编码性能提升70+%，**落地荣耀8**；单目HDR在麒麟650平台，性能提升17%，从1500ms下降到1250ms，在8953平台性能提升1倍，从2600ms下降到1300ms，内存减少47%，**落地G9、麦芒5**等上市产品;

2） **提供第三方兼容性工具检测**，提前识别由于N版本切换的不兼容的情况，并提前进行整改，**协助解除N版本的风险**；**提供KSAN工具**，帮助**识别有效问题10+个**，有效提升了终端产品的用户体验

2016年下半年进展和计划：

1）旗舰Mate9双目HDR 耗时：3+3模式**提升39.3%，890->540ms；5+1模式提升43.5%, 1150->650ms；内存：153->60M**

2） 旗舰Meta9deblur 耗时：**377ms->280ms**

自研编译器的未来

LLVM当前现状：

1）社区状态：当前LLVM社区非常活跃，平均3-5个月不等，release一个版本，社区活跃度非常高，当前根据实测，在ARM64的架构上，LLVM的性能已经略优于GCC的性能。

2）Lab业务定位：LLVM有着非常好的解耦能力，能够支撑跨平台迁移，可伸缩的空间非常大，向下往编译器延伸，向上可扩展编程语言，与Lab业务匹配度非常高；

3）产品诉求和公司当前现状：需要解决开放式编译器，解决芯片外销的问题（BSD license）。终端需要开放式编译器问题，解决工具集成（Clang的代码提示）、质量提升（静态及动态代码纠错）、社区同步（Android已全线切换至LLVM）等问题。IT产品线及固网需要解决开放式语言的编译执行（P4，SQL，JIT）及pipeline结构的优化及资源排布问题。

异构并行当前现状：

1）业界状态：当前异构框架和语言层出不穷，OpenVX、OpenCL、Halide等非常活跃，业界大公司都在构建异构能力。（高通/ARM：OpenCL/OpenGLES、Google：Halide/RenderScript/TPU、Nvidia：CUDA）

3）产品诉求和公司当前现状： 异构适用于数据密集以及性能要求高的应用场景，OpenCL框架定制化的调度系统硬化到SoC芯片；终端产品Camera后处理场景，当前通过手工优化，是一个异构加速的场景；未来智慧手机，对这个需求更强烈。

- 2016能力储备，OpenCL, OpenVX, Halide
- 2017构建平台，语言级异构，runtime异构
- 2018 面向智慧系统的高性能底座

异构加速能力构建思路：

先探索当前可能的异构框架平台特征，考虑探索过程中的更优手工算法优化落地，逐步构建平台能力。

LLVM能力构建思路：

从实际产品诉求出发，并考虑沿途下蛋，最终构建公司级LLVM能力中心。

对外合作及人才需求

- 智能计算IR设计(XLA/TVM/NNVM)
- 传统编译优化→基于算法语言+自动搜索的智能优化(TACO/ Spiral /Halide/Tangram/OpenTuner
- DSP DMA编程的抽象和简化

人才需求

- C++及开源编译器(GCC/LLVM)的高级人才
- 语言虚拟机的系统分析员、开发工程师
- 极致动手能力，计算机科学基础深厚
- 特别感谢中国科学院计算技术研究所编译器教研室多年来对华为编译器输送高素质人才，为自研编译器发展做出了很大贡献。

Thank you

www.huawei.com