DSP 应用中编译选项的智能选择

作者：CEVA 公司编译器项目经理 Eran Balaish

摘要

随着 DSP 处理器的能力越来越强大，可采用 C 编译器的代码部分在不断增加。不过，没有编程人员的协助，编译器是无法生成最优化的代码。为了最大地提高性能，编程人员必须利用各种编译选项功能来调节编译器。

不幸的是，在 DSP 应用里，没有充分利用编译器调节能力的现象相当普遍。整个应用过程中，往往只利用一组相同的编译选项来进行编译。这种方法忽略了每一项功能的特殊需求。

编译选项的智能选择可以大幅提高性能，例如可以大大减少代码量。在评估产品成本时，代码量通常是主要考虑因素，因为这对所需存储量有直接的影响。本文将介绍如何减少代码的使用量以及其它重要资源的消耗量。

在指令周期数和代码量之间进行权衡，满足功能级的特殊要求

为了更好地了解如何利用编译选项的智能选择来节省代码量，必须熟悉指令周期数与代码量之间的权衡取舍，这方面的实例是循环展开 (loop-unrolling) 和软件流水 (SWP) 的常用编译器优化技术。

这种技术通过循环体的复制来进行循环展开，利用从循环语句内部到循环语句外部的某些指令的拷贝来实现 SWP。当使用带有深度管线的多事件超长指令字 (VLIW) 处理器时，该技术显得非常有用。这时，SWP 可中断循环语句内部的许多依赖关系，而循环展开能够大幅提高指令间并行性 (ILP)。

这里给出了简单的乘法与累加 (mac) 循环语句，以阐释循环展开和 SWP。

```
for (i = 0; i < 960; i++)
{
   sum += src1[i] * src2[i];
}
```

图 1. 基本的 C 语言级 mac 循环语句

下面摘选了一段效率相对低的简单汇编代码，是在为代码量最小化而调节 CEVA-X1641™ DSP 内核编译器时产生的。它使用了 SWP，但没有循环展开。这个循环语句的执行需要使用大约 959*2 = 1918 个指令周期，所需代码量小于 20 个字节。

```
LS0.ld {w}  (r3)+#2, a3l
LS0.ld {w}  (r4)+#2, a2l
SQ.bkrep {ds1}  #958
nop
{
    nop
    A.S.mac a2l, a3l, a0
 || LS0.ld {w}  (r3)+#2, a3l
 || LS1.ld {w}  (r4)+#2, a2l
}
```

图 2. CEVA-X1641 编译器产生的 mac 循环语句中的简单但效率较低的汇编实现方案

相反地，当为指令周期数最小化而调节时，CEVA-X1641 编译器产生的结果截然不同。在下面的例子中，可以看到 CEVA-X1641 编译器处于满功率状态，充分利用了 CEVA-X1641 Quad-Mac 架构提供的全部硬件。在每一个周期中，所有四个 mac 单元和整个 128 位存储带宽都同时使用。编译器展开循环语句 8 次，并充分利用 SWP。该循环语句的执行需要用大约 119*2 = 238 个指令周期，所需代码量超过 100 多字节。这比前面的循环实现方案快了 8 倍，但代价是代码量增加到 5 倍。

```
   LS0.ld {2dw}  (r3)+#8, a5, a7
   SQ.bkrep {ds1}  #118
|| LS1.ld {2dw}  (r4)+#8, a4, a6
|| LS0.ld {2dw}  (r3)+#8, a17, a3
   LS1.ld {2dw}  (r4)+#8, a16, a2
{
    A.M1.mpy a6l, a7l
 || A.M0.mac3 a6h, a7h, a18
 || A.L.mpy a4l, a5l
 || A.S.mac3 a4h, a5h, a0
 || LS0.ld {2dw}  (r3)+#8, a5, a7
 || LS1.ld {2dw}  (r4)+#8, a4, a6

    A.M1.mpy a2l, a3l
 || A.M0.mac3 a2h, a3h, a18
 || A.L.mpy a16l, a17l
 || A.S.mac3 a16h, a17h, a0
```

```
  || LS0.ld {2dw}  (r3)+#8, a17, a3
  || LS1.ld {2dw}  (r4)+#8, a16, a2
}

    A.M1.mpy a6l, a7l
||  A.M0.mac3 a6h, a7h, a18
||  A.L.mpy a4l, a5l
||  A.S.mac3 a4h, a5h, a0

    A.L.mpy a16l, a17l
||  A.S.mac3 a16h, a17h, a0
||  A.M1.mpy a2l, a3l
||  A.M0.mac3 a2h, a3h, a18
```

图3. CEVA-X1641 编译器产生的 mac 循环语句中的效率最优化但代码量较大的汇编实现方案

ILP 的提高将显著提升性能。不过，上面提到的大量代码复制可能需要额外的存储空间，这也许会超过正常的嵌入式应用限制。那么如何能够在提高性能的同时又不必增加存储容量呢？很快我们就会找到解决办法。但有一件事是确定的，在许多嵌入式应用中，指令周期数的增加通常意味着代码量的增加，反之亦然。

分析器 -- 大型应用中完成优化工作的最重要工具

在大型应用的优化中，好的分析器必不可少。它可以提供有关每个功能非常有用的信息，比如指令周期数、代码量、调用次数乃至存储冲突、缓存未命中 (cache misses) 等缓存相关信息。

尽管缓存相关信息对存储管理非常有用，但在选择编译选项时却用处不大。对我们的需求来说，最有用的参数是指令周期数和代码量。代码量是静态计算的，而指令周期数则是运行时获得的。

下面是 CEVA 的 SmartNcode™ Profiler 分析器提供的分析结果实例。被分析的应用是 AMR-NB (自适应多速率 – 窄带) 语音编解码器。

Funcion	Size	Calls	ExclTotal	ExclAv	ExclMin	ExclMax
search_3i40	1530	32	643328	20104	20092	20132
cor_h	1380	32	296128	9254	9254	9254
c_fft	806	16	184416	11526	11526	11526
Syn_filt	730	192	161216	840	835	849
Qua_gain	1038	32	147392	4606	4606	4606
Convolve	320	64	139200	2175	2175	2175

图4. 分析结果实例

‘Excl’’代表每功能专用的指令周期数，意指只考虑该功能自身消耗的指令周期数，而没有包括它调用的其它功能消耗的指令周期。综合信息也可以获得，但对我们的目标来说用处不大。Av、Min 和 Max 分别表示平均、最小和最大。

如何根据分析信息选择最佳编译选项

首先，需要选中一组编译选项以供选择。在流程末端每一个源文件都利用这组选项中的一个进行编译。例如，在 CEVA-X1641 编译器中，这一组选项可以如下：{-O3, -O4, -O3 -Os1, -O3 -Os2, -O3 -Os3, -O3 -Os4}，这里 -OX 为指令周期优化选项，而 -OsY 是代码量优化选项。X 或 Y 因子越大，优化程度就越高。

一旦最优化的选项组被确定，就对每一个选项执行下列步骤：

1. 构建应用。
2. 设置分析器以收集运行时间信息并执行应用。
3. 存储分析信息以供稍后分析之用。

若有了关于每一个编译选项的分析信息，就可以利用电子表格把所有的结果存储在一个表中。

Compilation option		O4		O3 Os1		O3 Os2	
function	calls	code size	cycle count	code size	cycle count	code size	cycle count
search_3i40	32	1582	771808	1518	783264	1160	797632
cor_h	32	1410	303552	646	304992	460	310528
c_fft	16	852	187024	616	185360	540	187648
Syn_filt	192	762	162016	396	236640	286	235584
Qua_gain	32	1054	155392	890	157248	808	161536
Convolve	64	328	141696	238	157600	214	158240

图5. 在汇总表中分析所有编译选项的信息

现在可以利用设置电子表格特性的规则，根据相关分析信息自动为每项功能选择编译选项。例如，可以决定在规定的代码量定界符 (比如 1KB) 内为每一项功能选择最快的选项。这样，最大的总代码量就是设置的定界符乘以应用中的功能数目。另一个可行方案是把指令周期数最少的选项用于那些需要消耗一定数量的总指令周期数 (如大于 1%) 的功能，而把代码量最小的选项用于其余的功能。CEVA 的 SmartNcode 工具链使用的是嵌入了先进调节能力的更加复杂的规则。这些调节能力可以根据应用的特定要求让这种考虑规则偏向代码量胜于指令周期数，反之亦可。

如何利用标准工具链配置带有所选编译选项的项目

每一项功能有最适合的编译选项是不够的。标准工具链只支持文件级上而非功能级上的编译选项指定。这就直接导致下列问题：如果同一个文件中有几项功能，我们应该做什么？每一个功能是否需要不同的编译选项？对此有几个可行的解决方案：

1. 使用 CEVA 的每功能编译选项设置 SmartNcode 功能。这种功能实际上可让用户忽略上述问题，因为它不再存在了。该方案可以与上一级的选择保持一致。
2. 以某种形式重新安排应用，以便于利用相同的编译选项对一个给定文件中的所有功能进行编译。换言之，利用适合于其所包含的全部功能的编译选项来编译每一个文件。类似于上一种方法，这也允许编程人员与上一级的选择保持一致。
3. 放弃或选择一个编译选项都是基于这一种设想：在同一个文件中有多项要求不同的功能。例如，如果单个文件中有两项功能，一个需要 -O3 -Os1 选项，另一个需要 -O3 -Os3 选项，则放弃 -O3 -Os2 选项就是很好的选择。在较为重要的情况中，应该检查涉及到的全部功能的所有分析结果，并基于此找出一个良好的折衷办法。这种方案不同于前两种，它不允许用户与上一级的选择保持一致。

对功能级设置编译选项的特殊支持

如果编译器支持功能级的编译选项，就不需要前节所述的规避方案了。CEVA 的 SmartNcode 工具链就提供这种支持。这种集成开发环境 (IDE) 为设置每功能编译选项提供用户友好图形界面。IDE 以 XML 文件的形式把这种信息传送给编译器。高级用户可以无需使用 IDE 直接编辑 XML 文件。

下图是设置每功能编译选项的用户界面。在演示项目的 demo.c 文件中有两项功能 — foo1 和 foo2：

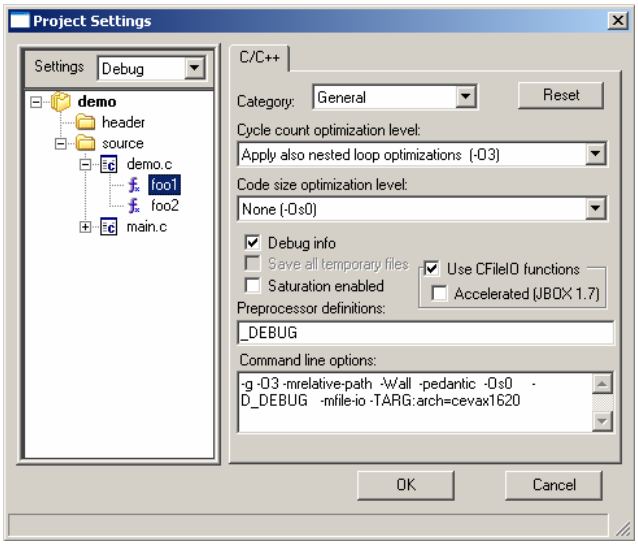

图 6. IDE 的每功能编译选项设置界面

下面是由 IDE 产生对应的 XML 文件，可把编译选项传送给编译器：

```
<File Name="demo.c">
  <Function Name="foo1" ChosenOpt="User">
    <Optimization Name="User" CodeSize="0" CycleCount="268516555">
      <Switch Name="-O3" IsComplex=""/>
    </Optimization>
  </Function>
  <Function Name="foo2" ChosenOpt="User">
    <Optimization Name="User" CodeSize="1" CycleCount="268516555">
      <Switch Name="-O4" IsComplex=""/>
    </Optimization>
  </Function>
</File>
```

图 7. 为编译器指定每功能编译选项的 XML 文件

集成化 — 可执行整个流程的自动化工具

这里描述的整个流程都由 CEVA 的 SmartNcode Project Optimizer 完全自动执行。这个 Project Optimizer 只需点击即可进行编译选项的智能选择。它利用每一个选项来构建应用并运行，使用分析器来取回有关每一项功能的信息，并为每一项功能选择最佳编译选项。然后建立一个带有所选编译选项的新配置结构。在整个流程中有一个很有用的向导 (wizard) 可帮助用户，如下所示：

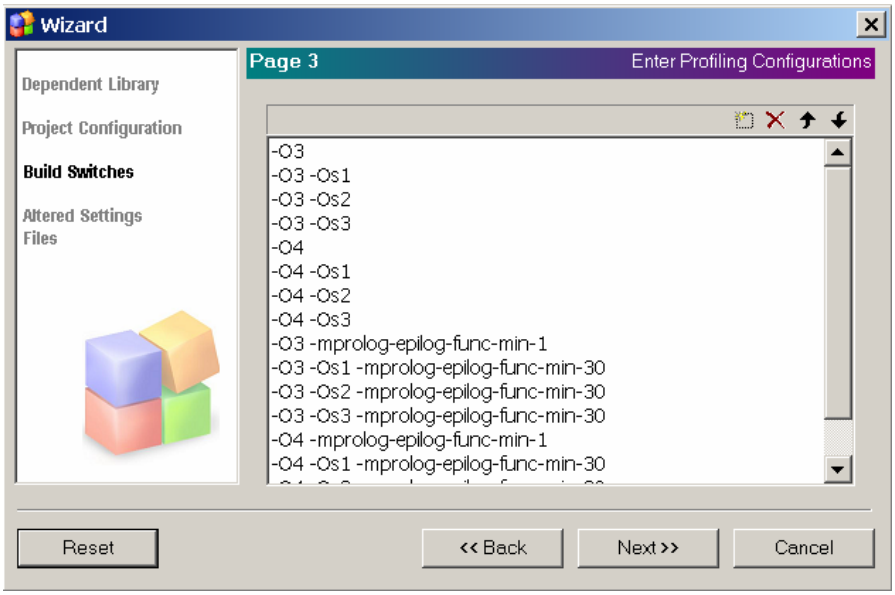

图8. Project Optimizer 向导

Project Optimizer 可在整个优化流程中提供有用的信息，并在各个阶段允许用户手动干预。例如，不论自动选择与否，用户都可以在某个特殊文件中强加一个编译选项。下表给出了所有分析结果。

opt.lnk | opt_Release.bld | main.c | Sncide1 - res | result2.opt

Projectes	Files	Functions	-O3 -Os1		-O3 -Os2	
			Code Size	Cycle Count	Code Size	Cycle Count
AMR_760_tes	q_gain_p.c	q_gain_pitch	468	0	376	0
	sp_dec.c	Speech_Deco	270	0	232	0
		Speech_Deco	64	0	64	0
	b_cn_cod.c	build_CN_para	278	0	150	0
		pseudonoise	184	0	112	0
		build_CN_code	318	0	242	0
	c1035pf.c	build_code3	884	0	786	0
		q_p	58	0	58	0
		code_10i40_35	390	0	310	0
	d2_11pf.c	decode_2i40_1	272	0	272	0
	d8_31pf.c	dec_8i40_31bit	280	0	252	0
		decompress10	206	0	208	0
		decompress_c	332	0	304	0
	c3_14pf.h	code_3i40_14b	822	4076	462	3995
	r_fft.c	c_fft	616	185360	540	187648
		r_fft	288	11696	234	11744
	dec_amr_aux.	Decoder_amr_	522	0	506	0
		Decoder_amr_i	218	0	212	0
	bits2prm.c	Bits2prm	178	0	144	0

图9. Project Optimizer 提供的分析信息

除给出所有所需信息之外，Project Optimizer 还可以显示对结果进行总结并带有几个可选性能点的图形。该图是交互式的，允许用户根据每一个点显示的结果来准确选择需要的项目配置。

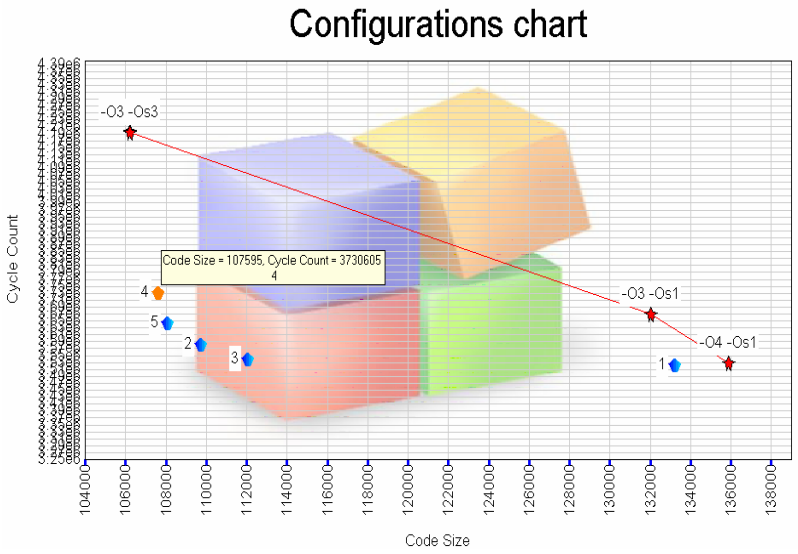

图 10. 显示可选性能点的交互式图形

这种方法在 AMR-NB 应用中的运用

AMR-NB (自适应多速率窄带) 是可实现声音合成器 (语音分析器和合成器) 的典型 DSP 应用。之所以选择 AMR-NB 是因为它是一种开源应用，这样可以自由展示在优化流程中获得的结果及结论。并且使用 CEVA-X1620™ SmartNcode 工具链来进行优化。

首先，AMR 密集使用 ETSI/ITU 标准中的基本操作，ETSI/ITU 标准定义了常用 DSP 操作的准确特性，比如加、乘和 MAC。这些基本操作在声音合成器中相当常见，故 CEVA-X1620 编译器以编译器固有特性的形式内建支持这些操作。第一阶段是实现编译器对 ETSI/ITU 基本操作的集成式支持。只需简单地用专用 SmartNcode 头文件替代某个应用头文件就可以非常容易地做到这一点。单这一个步骤就可以让应用的运行速度比开箱性能快 10 倍。

下一步是使用我们的方法来对编译选项进行智能选择。结果如下所示。采用了四个不同的优化级别：-O4、-O3 -Os1、-O3 -Os2 和 -O3 -Os3。图中四个红色的小点表示当对整个应用只运用一个选项时，该选项的性能。

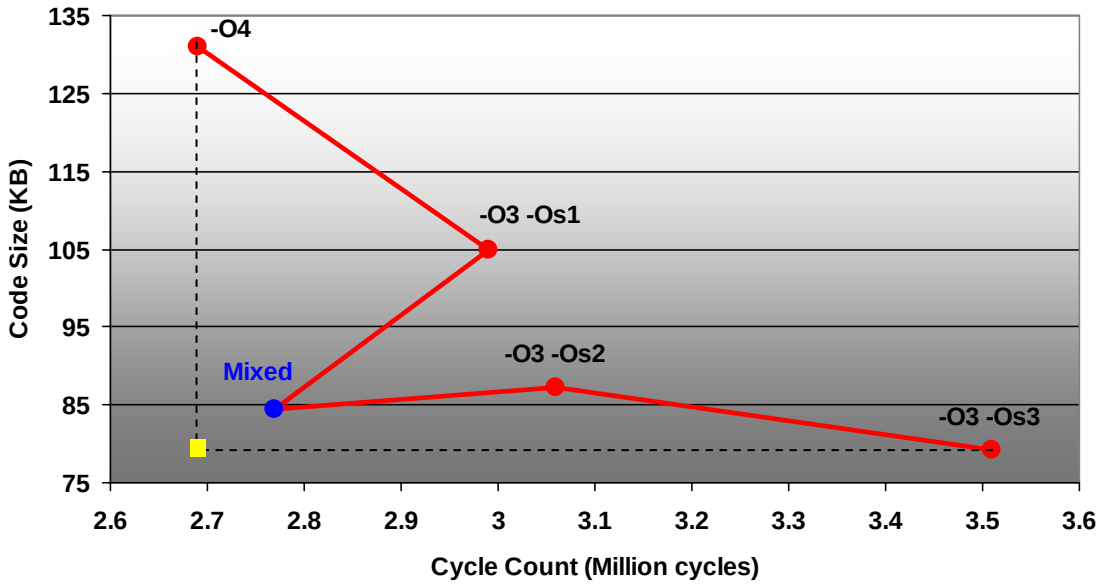

图 11. 在 AMR-NB 应用中运用这种方法的结果图示

这四个红点反映出前面描述在代码量和指令周期之间的权衡。-O4 选项最节省指令周期数，所需指令周期不到 270 万个，但它的代码量最大，超过 130 KB。-O3 -Os3 是最简单紧凑的选项，其代码量小于 80 KB，但同时也是最慢的，指令周期超过 350 万。-O3 -Os1 和 -O3 -Os2 选项介于 -O4 与 -O3 -Os3 之间。

图中黄色的小方块代表最佳性能结果，它是最佳指令周期数 (-O4 获得) 和最佳代码量 (-O3 -Os3 获得) 的综合。然而，根据代码量和指令周期数的权衡，这只是一个理论点，实际应用中是不可能获得的。图中最接近这个理论最佳点是标为“混合”的蓝点。混合代表编译选项的混合，不同于单个编译选项用于整个应用的情况。这也是我们利用编译选项智能选择方法获得的点。

‘混合’点清楚地显示了这种方法的优势。它的速度几乎和最快的选项相同，而指令周期数只稍大于 275 万；另一方面，它的简洁性又堪比最简单的选项，代码量仅略超过 84 KB。混合点显然是上述 5 种实际选项中的最佳选项。即使应用必需满足更少的指令周期目标，-O4 选项也不见得优于混合点选项，因为前者需要额外的代码量，这可能因多种存储冲突和缓存未命中而导致性能严重下降。

小结

总而言之，相比只利用单一选项，采用本文描述的方法选择的编译选项可以提供更好的结果。编译选项的智能选择可以把所需的代码量减至最少，从而直接减少代码量，间接减少指令周期数。

这种方法还可用来节省其它“昂贵的”资源。例如可把堆栈大小降至最小，假设编译器的编译选项影响功能级的堆栈空间消耗量。一般而言，这种方法适用于任何对应用的资源消耗有显著影响的相关编译选项。