无名稿件编号（由编辑插入）

通过文法演化优化嵌入式系统的 L1 缓存

´Josefa D´ıaz Alvarez · J. Manuel Colmenar · Jos´eL.Risco-Mart´ın · Juan Lanchares · Oscar Garnica

arXiv:2303.03338v1 [cs.NE] 6 Mar 2023

收到日期：日期 / 接受日期：日期

摘要 如今，嵌入式系统配备的缓存存储器容量已足够大，因而能够对系统性能和能耗产生前所未有的影响。此外，缓存系统已被确定为一种可根据正在运行的应用程序的内存访问模式调整配置，从而改善这些指标的组件。然而，鉴于缓存存储器有许多参数，且每个参数都可能设置为大量不同的值，设计人员面临着庞大且耗时的探索空间。

本文提出了一种基于语法进化（GE）的优化框架，能够为给定的一组基准测试应用程序高效地找到最佳缓存配置。这种元启发式方法能够大幅缩短优化运行时间，并在较少的代数内取得良好结果。此外，通过高效存储已评估的缓存，优化运行时间也得以进一步缩短。此外，我们选择语法进化，是因为语法具有灵活性，便于创建表型，从而构成对缓存模拟器的调用，而该模拟器是评估不同配置所必需的。

Mediabench套件的实验结果表明，与真实世界的基准配置相比，我们的方案能够找到使性能平均提升62%的缓存配置。

1 引言

在过去几十年里，嵌入式计算系统取得了长足发展，成为推动技术进步的关键力量之一。智能手机、数码相机、全球定位系统等便携式设备不断增加新功能。这些新功能大多旨在支持新的服务和多媒体应用，尽管嵌入式系统的资源有限。以便携式系统为例，它们依靠电池供电，而受设计限制，电池的容量和大小有限。此外，这些系统的散热能力有限，因此低能耗是一项主要要求。另一方面，便携式系统通常运行多媒体应用，这类应用要求高性能，因此能耗较高。因此，嵌入式系统设计人员面临的主要任务之一，就是以较低成本，在提升性能与降低能耗之间寻求恰当的平衡。

以往研究指出，片上缓存存储器是能耗最高的组件之一，在嵌入式处理器中，其能耗占芯片总功率的 20% 至 30% [31]。此外，缓存存储器的设计会显著影响能耗和性能，因为不同的设计会带来不同程度的硬件复杂性。例如，直接映射缓存设计的命中时间最快，

´Josefa D´ıaz Alvarez 埃斯特雷马杜拉大学梅里达大学中心，06800 梅里达，西班牙 电话：+34-924-387268 电子邮件：mjdiaz@unex.es

J. Manuel Colmenar 计算机科学与统计系，胡安·卡洛斯国王大学，28933莫斯托莱斯，西班牙 电子邮箱：josemanuel.colmenar@urjc.es

Jos´e L. Risco-Mart´ın · Juan Lanchares · ´Oscar Garnica 计算机体系结构与自动化系，马德里康普顿斯大学，28040马德里，西班牙 电子邮箱：{jlrisco,julandan,ogarnica}@dacya.ucm.es

但与 N 路组相联缓存相比，要获得相当的性能，就需要更大的缓存大小。另一方面，组相联缓存的能耗较高，因为每次操作都会访问一组 N 个标签 [15]。

缓存存储器的行为不仅受结构参数（容量、块大小、相联度等）的影响，还受搜索、预取和替换算法、写策略等参数的影响。所有这些特性共同构成了所谓的缓存配置。为这些参数找到最优值，将带来最佳性能和最低能耗，从而得到最佳缓存配置。

此外，不同应用程序的行为各异，因此需要不同的缓存来满足其在能耗和性能方面的特定目标。因此，我们需要考虑目标应用程序的行为，以找到能够提升性能或降低能耗的最佳缓存配置。

总之，用于确定最佳缓存配置的优化方案应当：（1）遍历缓存配置的设计空间，该空间定义为所有缓存参数取值的各种组合；（2）针对每种候选缓存配置，考虑目标应用程序的能耗和性能，因为其内存访问模式各不相同。

评估缓存配置需要使用缓存模拟器处理目标程序先前执行时的缓存访问跟踪记录。通过这一过程，可以计算所研究应用程序的执行时间和能耗。这一过程速度较慢，因为程序跟踪通常会记录数十亿次内存操作，模拟器运行这些记录需要数十秒。因此，考虑到可定义的缓存配置数量众多，对该设计空间进行穷举搜索将耗费难以承受的时间。

另一方面，启发式技术在搜索最优解的过程中能够减少评估次数，因此非常适用于这类问题。因此，本文提出了一种基于语法进化（GE）的优化方案，[11] 该方案利用程序跟踪，为给定的目标应用程序获取在执行时间和能耗方面经过优化的缓存配置。与穷举方法相比，执行时间大幅缩短，原因有二：（1）元启发式算法即使在代数和种群规模较小的情况下也能收敛；（2）我们添加了一个哈希映射，用于存储每个已评估缓存的目标值。此外，GE 中语法的灵活性有助于生成缓存模拟器每次评估所需的参数，从而使其更容易与优化算法进行通信，使其更易于与优化算法进行通信。

为了检验我们的方案，我们针对一组取自 Mediabench 基准测试集 [18],的多媒体应用程序进行了实验，以寻找最佳缓存配置。这些应用程序具有代表性，涵盖图像、音频和视频处理，这些都是嵌入式系统的典型应用程序。实验中，我们假设采用基于 ARM9 处理器系列 [2], 的硬件架构，该系列广泛应用于多媒体嵌入式设备。后文将展示，与基准缓存相比，优化后的缓存配置平均可将执行时间和能耗分别降低 75% 和 96%。此外，优化运行时间也在可接受范围内；我们估计，与未存储评估次数的语法进化实现相比，优化运行时间平均减少了 94%。

本文其余部分安排如下。第 2 节介绍缓存设计的主要问题，并定义该问题的搜索空间。第 3 节回顾缓存优化技术的相关研究。第 4 节介绍本研究采用的性能与能耗模型，用于计算候选解的优劣。第 5 节介绍整个优化框架，详细说明离线流程以及我们采用的语法进化方法。随后，第 6 节分析实验结果。最后，第 7 节总结全文并展望未来工作。

2 缓存设计与搜索空间

为明确所研究问题的范围，本节将介绍一些缓存设计概念及相关问题。这些概念也有助于我们确定搜索空间的大小。

首先需要注意的是，存在不同的设计特征（也称设计参数），需要对其进行调优，以确定缓存配置。赋予这些参数的取值会影响缓存行为，包括执行时间和能耗。不过，这些参数可能因缓存所集成的处理器不同而有所差异。

在这方面，我们首先必须决定为本研究选择哪种类型的缓存。考虑到我们的方法聚焦于嵌入式系统，我们选择了 ARM9 处理器系列的缓存 [2]。

ARM处理器因功耗低、成本低，在便携式设备市场中占据主导地位，并广泛应用于多媒体嵌入式设备。事实上，Apple A7,

热门的 iPhone 5S、iPhone 6 和 iPad Air 2 所搭载的 A8 和 A8X 处理器确实实现了 ARMv8-A 指令集。

此外，ARM9 处理器系列还应用于多种设备，例如游戏机（GP32、任天堂 DS）、计算器（HP 49/50）、手机（HTC TyTN、索尼爱立信 K 和 W 系列）以及车载全球定位系统设备（三星 S3C2410）。

属于这一系列的处理器支持安装大小介于 4 KB 至 128 KB 的缓存存储器。例如，ARM920T 、ARM922T 和 ARM940T 处理器（ARM9TDMI 系列）分别采用独立的 16 KB、8 KB 和 4 KB 指令缓存与数据缓存。尽管 ARM9E 系列的后续版本支持最高达 1 MB 的缓存大小，例如 ARM946E-S 处理器。

鉴于较新的 ARM9 处理器普遍采用数据缓存和指令缓存分离的设计，我们的工作也考虑了这种双缓存方案。

因此，该处理器的指令缓存和数据缓存中可调优的缓存参数如下：指令缓存和数据缓存的缓存大小、缓存行大小、替换算法、相联度和预取算法，以及数据缓存的写策略。缓存大小是缓存存储器的容量，以字节为单位；块大小表示每次缓存访问所读取或写入的数据量；替换算法负责决定从缓存存储器中淘汰哪个块；预取算法则决定如何将块载入缓存存储器。最后，写策略决定缓存中存储的数据何时写入主存，且仅影响数据缓存。

简而言之，我们要解决的问题是为缓存参数选择合适的取值，以优化给定应用程序的执行时间和能耗。因此，搜索空间由各个缓存参数不同取值的所有组合构成。在我们的目标系统中，缓存配置由11个参数组成：指令缓存占5个，数据缓存占6个。根据嵌入式系统中典型的缓存配置，我们为每个参数选定了允许取值范围，如图1所示。

对于指令缓存（I-Cache），缓存大小可取 8 种不同的值，范围介于 512 字节和 64 KB 之间；块大小可取 4 种不同的值，可选择 3 种替换算法和 3 种预取算法，

并且有 8 种相联度可用。因此，指令缓存共有 2304 种可能的配置。

数据缓存（D-Cache）具有与指令缓存相同的参数，但增加了 2 种可能的写策略

。因此，可以选择 4608 种可能的数据缓存配置。

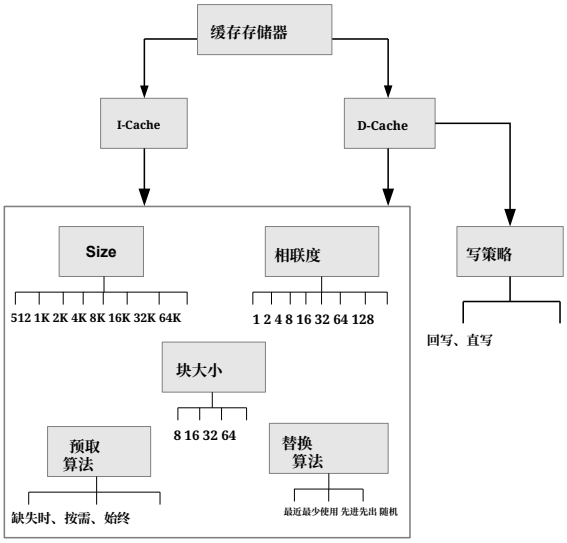

图1 缓存配置参数。指令缓存和数据缓存都必须使用可用值进行定制。

因此，考虑两个缓存的所有可能组合，搜索空间的大小为 $2304 \times 4608 = 10616832$ 种配置。然而，其中一些约束与不可行的参数组合有关。例如，512B、64路组相联且块大小为32字节的缓存配置是不可行的。因此，如果为指令缓存或数据缓存选择了这一组合，则必须由任何优化算法将其判定为无效。

3 相关工作

众所周知，缓存存储器是对提升性能和降低能耗起着关键作用的组件。近几十年来，许多研究人员一直致力于改进缓存存储器的不同特性。人们提出了多种优化方法，以提升性能和/或降低能耗、开发仿真和评估工具等。接下来，我们将分析一些与本文提案相关的工作，并将其分为两类：（1）涉及缓存重配置的工作；（2）基于进化技术的工作。

3.1 缓存重配置

多年来，主要目标之一是根据正在运行的工作负载配置缓存存储器，

以主要提升性能和/或降低能耗。这既需要硬件支持缓存重配置，也需要算法来确定参数的合适取值，从而结合正在运行的应用程序实现最佳性能。因此，许多研究工作从不同角度分析了缓存存储器参数，并提出了新的重配置技术。

Naz 等人 [20] 提出了一种面向嵌入式系统的分离式数据缓存设计。数据缓存被划分为标量缓存和数组缓存，分别旨在利用时间局部性和空间局部性。他们的优化基于预定义的缓存配置，其中各参数值在优化开始前已固定。因此，与考虑所有参数值的搜索空间相比，这种情况下的搜索空间较小。

Chen 和 Zou [8] 提出了一种适用于嵌入式系统的高效重配置管理算法。识别出阶段变化后，缓存配置会自动改变。搜索空间由以下参数构成：缓存大小、块大小和相联度。在我们看来，这项工作考虑的缓存参数数量不足。

Gordon-Ross 等人 [14] 采用了一种结合离线阶段分类和在线阶段预测器的方法
阶段分类将应用程序的执行划分为固定大小的时间间隔，并根据相似的行为模式对这些间隔进行分组。阶段预测器会决定下一个时间间隔采用的缓存配置。探索空间由缓存大小、块大小和相联度定义，也就是说，搜索空间仅由这三个参数构成。在这种情况下，由于需要在线预测阶段，因此所需参数较少。

Wang 等人 [33]提出了面向实时嵌入式系统的动态缓存重配置。他们通过在运行时执行静态分析来降低能耗。这项方案基于先前的研究 [32,34]，分别探讨了统一的两级缓存层次结构和多级缓存层次结构。这两种情况下都结合了静态分析与缓存参数动态调优，且不会产生时间开销。不过，优化的参数数量较少：缓存大小（1、2 或 4KB）、缓存行大小（16、32 或 64 字节）以及相联度（1 路、2 路或 4 路）。

新的硬件技术和基于核心的处理器技术（例如 ARM [3], 提出的技术）允许针对每个应用程序更改缓存配置。这些更改会影响几个主要参数：容量、块大小和相联度。然而，需要一种高效算法来确定每个应用程序的最优值。

3.2 进化技术

进化技术已广泛应用于设计优化，包括针对不同目标的硬件优化和软件优化。

语法进化技术的创作者发表了一项工作，自动生成缓存算法 [22]。然而，他们只考虑了缓存算法的生成，并将其他一些参数（例如缓存大小）设为固定值。此外，他们还遗漏了许多参数。

Palesi 和 Givargis [25] 提出了一种多目标优化方法，用于探索片上系统（SoC）架构的设计空间，以找到系统的帕累托最优配置。他们通过遗传算法执行优化，评估则直接在 SoC 上进行。然而，他们研究的参数数量少于我们的方案，而且由于 SoC 性能较低，他们的方法无法扩展到当前的多媒体应用程序。

Filho 等人 [30] 提出了一种基于 NSGA-II 算法的方法，用于评估二级缓存的缓存配置，以改善能耗和性能，并对缓存大小、缓存行大小和相联度进行优化。然而，他们考虑的参数数量同样少于我们的方案，因此搜索空间也更小。

Dani 等人 [9]采用遗传算法，为芯片多处理器（CMP）系统寻找最优缓存配置。他们处理的是 CMP 这类复杂系统。然而，他们没有管理更多的缓存参数；而且，由于他们将芯片和缓存的配置编码到一个染色体中，解码过程较为复杂。这一问题使得相同方法难以应用于不同的微架构。

Risco 等人 [28] 提出了一个基于语法进化的框架，用于为嵌入式系统中的多媒体应用程序设计定制的动态内存管理器。他们优化了内存访问、内存使用和能耗。然而，他们并未处理缓存参数，而是优化操作系统内存管理器，并未涉及缓存设计参数。

最后，我们深入查阅了相关文献，特别是语法进化出版物的主要集散地之一 [1], ，但没有发现近期有研究将这一技术应用于优化微架构的缓存设计。

3.3 总结

如前所述，采用重配置的方法会增加存储器子系统设计的硬件复杂度。此外，在大多数情况下，这种复杂度还会增加执行时间。我们的工作提出了一种不同的方法，旨在探索缓存设计的搜索空间后，找到最佳缓存配置。这种策略避免了缓存重配置，因此不会增加缓存存储器设计的硬件复杂度。

此外，先前的研究通过指令缓存或数据缓存的缓存大小、块大小和相联度来定义其设计空间，但我们尚未发现有研究提出在嵌入式系统中同时考虑这两种缓存。我们的研究动机有所不同，因为我们同时考虑指令缓存和数据缓存的优化，并选取了所有可调参数：两种缓存的缓存大小、块大小、相联度、替换算法和预取算法，以及数据缓存的写策略。因此，我们探索了更广阔的搜索空间。

此外，据我们所知，先前的研究中没有一项像我们的方案这样同时优化如此多的参数。关于采用进化技术的研究，大多数被引用的论文都将空间探索集中在缓存大小、缓存行大小和相联度等典型的缓存结构参数上，而且通常只考虑较窄的参数取值范围。在这类研究中，最常用的技术之一——遗传算法——存在编码过程方面的缺陷：针对每种不同的缓存范式和模拟器软件，都必须对编码过程进行定制。

另一方面，我们提出的方法可应用于任何类型的缓存和处理器架构。为此，我们框架中唯一需要进行的主要更改是缓存模拟器和语法，后文将对此进行说明。

4 性能与能耗模型

在这项研究中，我们考察运行基准测试应用程序时特定缓存配置的执行时间和能耗。如前所述，我们处理由缓存模拟器处理的程序跟踪；模拟器会统计每次执行的命中次数和未命中次数，以此反映缓存行为。

如前所述，我们选择 ARM9 处理器系列 [2] 作为优化缓存配置的目标嵌入式系统。该系统的一级缓存分为指令缓存和数据缓存，并使用嵌入式动态随机存取存储器作为主存。

表 1 主存（动态随机存取存储器）特性。

Size	64 Mb
访问时间	3.9889×10^{-9} 秒。
带宽	6.7108864×10^{9} 字节/秒。
访问功耗	1.051 瓦

除目标架构外，我们还假设指令缓存为只读，并且存在主存。主存的特性列于表 1 中，并将在缓存配置评估中予以考虑。

硬件特性设定后，我们需要一个模型，将基准测试运行中的命中次数和未命中次数转换为执行时间和能耗。为此，我们在优化框架中采用了 [16] 所述的性能与能耗模型。

论文作者证明，这些模型与缓存模拟器得出的结果完全等效，准确率达到 100%。鉴于其性能，并考虑到模型表达式的计算速度很快，因此它们非常适合用于我们所介绍的这类优化算法。

因此，基于这些模型，我们将缓存存储器子系统的性能与能耗模型纳入考量。由于我们的优化方案仅针对缓存子系统，因此未考虑芯片其他组件（中央处理器、主存等）的模型。下面将简要介绍这些缓存模型。

4.1 性能模型

缓存性能模型可用于求得执行时间。该模型基于缓存系统中的命中次数与未命中次数，以及解决这些情况所需的时间。公式（1）展示了用于计算执行时间的方程。下文将介绍其中的各个组成部分，更详细的说明见 [16]。

$$
\begin{aligned}
T = {} & Icache_{access} \times Icache_{access_time} + \\
& Icache_{miss} \times DRAM_{access_time} + \\
& Icache_{miss} \times Icache_{line_size} \times \frac{1}{DRAM_bwidth} + \\
& Dcache_{access} \times Dcache_{access_time} + \\
& Dcache_{miss} \times DRAM_{access_time} + \\
& Dcache_{miss} \times Dcache_{line_size} \times \frac{1}{DRAM_bwidth}
\end{aligned}
\tag{1}
$$

方程中的每一项分别表示：

– $Icache_{access}$ 和 $Dcache_{access}$ 分别表示对指令缓存和数据缓存的访问次数。– $Icache_{miss}$ 和 $Dcache_{miss}$ 表示缓存未命中次数，每次发生未命中时，都必须从主存复制数据。– $Icache_{access\ time}$ 和 $Dcache_{access\ time}$ 分别表示访问指令缓存和数据缓存所需的时间。– $DRAM_{access\ time}$ 表示主存延迟时间。– $Icache_{line\ size}$ 和 $Dcache_{line\ size}$ 分别对应指令缓存和数据缓存的缓存行大小（块大小）。– $DRAM\ bwidth$ 表示动态随机存取存储器的传输容量

因此，该方程中的第一部分和第四部分分别由 $Icache_{access} \times Icache_{access\ time}$ 和 $Dcache_{access} \times Dcache_{access\ time}$ 表示，用于计算解决所有指令缓存访问和数据缓存访问所需的总时间。$Icache_{miss} \times DRAM_{access\ time}$ 和 $Dcache_{miss} \times DRAM_{access\ time}$ 是主存响应指令缓存未命中和数据缓存未命中所花费的总时间。第三部分和第六部分分别由 $Icache_{miss} \times Icache_{line\ size} \times \frac{1}{DRAM\ bwidth}$ 和 $Dcache_{miss} \times Dcache_{line\ size} \times \frac{1}{DRAM\ bwidth}$ 指定，用于计算发生未命中时填充指令缓存行和数据缓存行所需的总时间。

– $Icache_{access_energy}$ 和 $Dcache_{access\ energy}$ 分别对应每条指令和每次访问数据缓存的能耗。

$Icache_{access} \times Icache_{access\ energy}$ 和 $Dcache_{access} \times Dcache_{access\ energy}$ 项分别计算指令缓存和数据缓存的能耗。发生未命中时，$Icache_{miss} \times Icache_{access\ energy} \times Icache_{line\ size}$ 和 $Dcache_{miss} \times Dcache_{access\ energy} \times Dcache_{line\ size}$ 项表示向指令缓存和数据缓存写入数据所需的能耗成本。最后两项计算动态随机存取存储器响应缓存未命中时的能耗成本。

在我们的方法中，由于以下三个原因，我们移除了能耗方程中的第一项 $Total_{time} \times CPU_{power}$：(1) 项 CPU_{power} 为常数，而项 $Total_{time}$ 对应的是系统的总时间，并非仅指缓存子系统的时间；(2) 该项表示中央处理器消耗的能耗，而我们只优化存储器子系统的性能和能耗；(3) 该项可能大到足以掩盖缓存操作造成的差异。因此，能耗方程可简化为：

$$
\begin{aligned}
E = {} & Icache_{access} \times Icache_{access_energy} + \\
& Dcache_{access} \times Dcache_{access_energy} + \\
& Icache_{miss} \times Icache_{access_energy} \times Icache_{line_size} + \\
& Dcache_{miss} \times Dcache_{access_energy} \times Dcache_{line_size} + \\
& Icache_{miss} \times DRAM_{access_power} + \\
& (DRAM_{access_time} + Icache_{line_size} \times \frac{1}{DRAM_{bwidth}}) + \\
& Dcache_{miss} \times DRAM_{access_power} + \\
& (DRAM_{access_time} + Dcache_{line_size} \times \frac{1}{DRAM_{bwidth}}) \qquad (3)
\end{aligned}
$$

所有公式分别使用秒、瓦特、焦耳、字节和字节/秒作为时间、功率、能耗、缓存行大小和带宽的度量单位。

接下来，我们将介绍优化的完整过程，并说明这些模型的应用位置。

4.2 能耗模型

能耗模型定义于公式 (2) 中，并在[16]中进行了详细说明。该方程用于确定某一缓存配置的动态能耗。

$$
\begin{aligned}
E = {} & Total_{time} \times CPU_{power} + \\
& Icache_{access} \times Icache_{access_energy} + \\
& Dcache_{access} \times Dcache_{access_energy} + \\
& Icache_{miss} \times Icache_{access_energy} \times Icache_{line_size} + \\
& Dcache_{miss} \times Dcache_{access_energy} \times Dcache_{line_size} + \\
& Icache_{miss} \times DRAM_{access_power} + \\
& (DRAM_{access_time} + Icache_{line_size} \times \frac{1}{DRAM_{bwidth}}) + \\
& Dcache_{miss} \times DRAM_{access_power} + \\
& (DRAM_{access_time} + Dcache_{line_size} \times \frac{1}{DRAM_{bwidth}}) \qquad (2)
\end{aligned}
$$

上文已简要介绍了方程中的大多数组成部分。其余部分如下：

– $DRAM_{access\ power}$ 表示每次访问动态随机存取存储器的功率。

5 优化方法论

如前所述，在这项工作中，我们提出了一种方法，用于为给定的一组应用程序确定最佳缓存配置。我们将最佳缓存配置定义为能够使给定基准测试的执行时间和能耗最小化的配置。

为此，本节将介绍我们提出的优化框架，用于优化多媒体

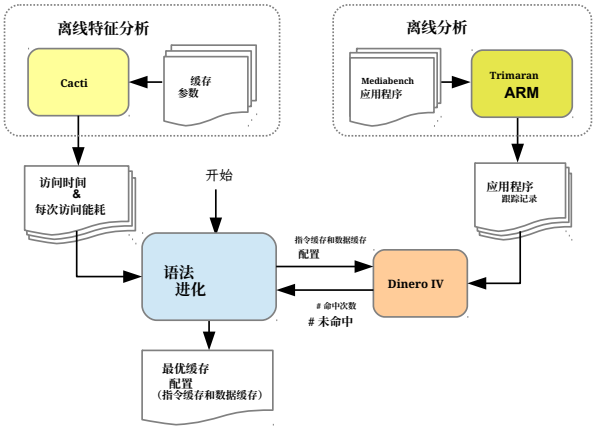

图 2 缓存配置优化中涉及的流程。

嵌入式系统的缓存存储器。图 2 展示了整个流程的概览。

我们将优化过程分为三个不同阶段。首先，在优化开始前只需运行两个流程各一次。这两个流程称为离线流程，分别执行：(1) 缓存特性分析和 (2) 应用程序分析。第三个阶段是优化，由运行语法进化的进化过程和调用缓存模拟器组成。

如图 2 所示，优化由语法进化（GE）模块执行。每当需要评估一种配置时，GE 都会调用 Dinero IV，这是一款跟踪驱动缓存模拟器 [12]。它接收 GE 提供的缓存配置，并返回目标应用程序跟踪记录中的缓存命中次数和未命中次数。这些数据会纳入模型，用于评估该配置。接下来，我们将更详细地介绍这一优化流程。

5.1 离线流程

离线运行两项任务：分析可能的缓存配置，以及分析目标基准测试。这两项任务都很耗时，但只需运行一次，因为其结果会存储在数据文件中，供缓存模拟器和 GE 模块读取。

图 2 左上部分所示的特征分析过程，具体任务是组合缓存配置的所有参数值。也就是说，它会生成所有可能的缓存配置，并获取其访问时间以及每次访问的能耗。因此，该过程提供了方程（1）和（3）中各项所用的时间和能耗值。为此，我们使用 Cacti [19]，并考虑 32 nm 工艺技术来计算动态随机存取存储器和缓存的访问时间

以及每次缓存访问所消耗的动态能耗。Cacti 是一种解析模型，广泛用于估算缓存存储器的能耗、功耗、性能和面积。该特征分析仅运行一次；我们选择了Cacti组织提供的网络工具来完成此任务。Cacti 所需的最重要参数主要与硬件相关：缓存大小、缓存行大小和相联度，而且指令缓存与数据缓存之间没有区别。因此，虽然这项任务耗时较长，但成本可以接受。

图2右上部分的性能分析过程负责从目标应用程序生成跟踪记录。同样，该过程只需运行一次。性能分析会处理每个基准测试应用程序，并从一次运行中获取缓存命中和未命中的序列，从而生成内存跟踪文件。此过程由名为Trimaran [7], 的开源编译器和性能监控基础设施执行，它提供了获取准确应用程序跟踪所需的资源。由于Trimaran无法直接处理ARM处理器，我们对其进行了修改，加入了架构模拟器SimpleScalar [5], 中的代码。SimpleScalar能够模拟多种不同的架构，其中包括ARM。

此外，还必须指出，所研究的所有基准测试的输入和输出数据都表示在跟踪文件中。因此，为了捕获每个应用程序的平均行为，我们将选择各基准测试文档中注明的推荐输入。

5.2 语法进化

缓存特征分析最终得出两类结果。一方面，获得了每种缓存配置下每次访问所需的时间和能耗；另一方面，获得了包含每个基准测试应用程序所有内存操作的跟踪记录。因此，我们的框架已准备就绪，可用于评估任何候选缓存配置的性能。这些配置将在基于语法进化的优化模块中生成。

语法进化 [23,24,21,11] 是一种进化技术，能够通过语法来表示个体。GE 是遗传编程（GP） [17] 的一种形式，它通过进化程序，并根据程序的适应度值对其进行评估。在这方面，GE 使用研究人员定义的语法所生成的表达式来表示每个程序，也就是每个个体。

语法进化对可变长度的二进制字符串执行进化过程，并通过映射生成程序

其方法是在巴科斯-诺尔范式（BNF）语法定义中选择产生式规则。由于这些字符串类似于染色体，语法进化可以使用选择、交叉和变异等遗传算子。金融建模 [4],游戏策略 [13],动态游戏环境 [26], 2D 平台游戏设计 [29], 三维设计 [6] 等问题，都已通过使用语法进化得到成功解决。

因此，语法进化会根据语法定义遍历缓存配置的搜索空间。对于每个个体，它都会调用缓存模拟器，以获取目标基准测试程序的命中次数和未命中次数。然后，利用此前由 Cacti 计算出的访问时间和每次访问能耗，以及第 4 节所述的性能模型和能耗模型，计算适应度。该过程会重复执行，直到达到为语法进化选定的代数。最终，将为每个基准测试程序获得一个经过优化的缓存配置。

如第 2 节所述，该问题的搜索空间由缓存配置集合定义。然而，语法进化使用表示这些配置的个体，并通过适当的编码对其进行表示。事实上，与遗传编程中的基于树的结构不同，语法进化使用线性基因组。线性基因组便于应用遗传算子。在这里，每个个体的染色体由一串对应于密码子的整数值组成。每个密码子存储一个 8位值，从而形成基因型。基因型在一个称为映射的过程中被解码，其中使用语法将密码子转换为个体的表型。

每个个体代表一种特定的缓存配置，必须对其进行评估，以测量其执行时间和能耗。在这方面，我们决定对缓存的这两项特征赋予相同的权重。因此，对于给定的缓存配置 c_i，评估过程通过适应度函数 f 进行，该函数定义于式（4）中。

$$f(c_i) = 0.5 \times \frac{T(c_i)}{T(Baseline)} + 0.5 \times \frac{E(c_i)}{E(Baseline)} \quad (4)$$

适应度的每个组成部分，即执行时间和能耗，均相对于基准配置进行归一化。通过以基准为参照进行归一化，我们能够正确地缩放并处理时间和能耗等不同度量的单位。如公式（4）所示，两项的权重均设为 50%，以求在性能提升与能耗之间取得最佳平衡。语法进化中的最佳缓存配置就是使适应度值最小的配置。

```
N = { <DineroParams>, <CacheSizeB>, <LineSizeB>,
      <ReplAlg>, <Assoc>, <PrefAlg>, <WritePol> }
T = { 1, 2, 4, 8, 16, 32, 64, 128 512 , 1024, 2048,
      4096, 8192, 16384, 32768, 65536,  l, f, r, m,
      d, a, n }
S = <DineroParams>

P = { I    <DineroParams> ::= -l1-isize <CacheSizeB>
                              -l1-ibsize <LineSizeB>
                              -l1-irepl <ReplAlg>
                              -l1-iassoc <Assoc>
                              -l1-ifetch <PrefAlg>
                              -l1-dsize <CacheSizeB>
                              -l1-dbsize <LineSizeB>
                              -l1-drepl <ReplAlg>
                              -l1-dassoc <Assoc>
                              -l1-dfetch <PrefAlg>
                              -l1-dwback <WritePol>

      II    <CacheSizeB> ::= 512 | 1024 | 2048 | 4096
                           | 8192 | 16384 | 32768
                           | 65536
      III   <LineSizeB> ::= 8 | 16 | 32 | 64
      IV    <ReplAlg> ::= l | f | r
      V     <Assoc> ::= 1 | 2 | 4 | 8 | 16 | 32 | 64
                      | 128
      VI    <PrefAlg> ::= m | d | a
      VII    <WritePol> ::= a | n  }
```

图3 缓存配置描述语法。

鉴于评估缓存配置需要调用缓存模拟器，我们设计了一种语法，用于生成作为模拟器调用参数的表型。事实上，语法进化的优势之一在于能够根据由整数值构成的基因型创建复杂的表型。

图3以巴科斯-诺尔范式格式展示了我们针对前文定义的搜索空间所设计的缓存优化语法。如前所述，该语法生成将发送给缓存模拟器的参数字符串。实际上，我们已根据模拟器的要求调整了参数。如图所示，替换算法将取值 l、f 或 r，在模拟器中分别对应最近最少使用、先进先出和随机。预取和写策略也同样如此。

按照术语定义，语法可表示为元组 $\{N, T, S, P\}$，其中N 表示非终结符，T 是终结符集合，S 表示属于N的起始符号，而P 定义了将N 中的值映射到T的产生式规则集合。“|”符号用于分隔单条产生式规则中的不同选项。在我们的语法中，P 由从 I 到 VII 的产生式规则构成。

接下来，我们使用图3中的语法，展示一个针对当前问题的映射过程示例。我们将图4上半部分所示的基因型视为一个候选解。这个个体是一个表示缓存配置的基因型。不过，为了获得缓存配置中每个

此前所述的 11 个参数的值，必须对其进行解码。

映射或解码过程通过对密码子值与对应于正在处理的非终结符的每条产生式规则的选项数量进行取模运算来完成。

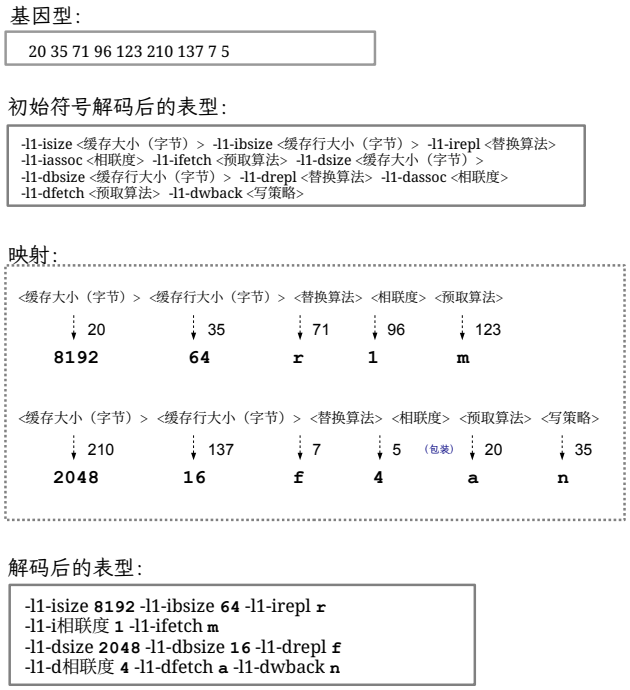

图4 语法进化的解码过程：从上方的基因型出发，通过所提出的语法进行映射，得到表型。

接下来，以示例个体为例，解码过程如图4所示。解码从起始符号 <DineroParams> 开始，该符号由其产生式替换，对应于对缓存模拟器的参数化调用。这里存在多个非终结符，第一个是 <CacheSizeB>。因此，选取基因型中的第一个基因，即 20。由于 <CacheSizeB> 的产生式规则（规则 II）有 8 种不同的选项，选定的值对应选项 4（$20\ MOD\ 8 = 4$），即 8192。请注意，该规则的第一个选项对应 0，第二个对应 1，依此类推。接着读取第二个基因，并用其值 35 解码下一个非终结符 <LineSizeB>。该符号的产生式规则为 III，共有 4 个选项。因此，经过取模运算（$35\ MOD\ 4 = 3$）后，<LineSizeB> 被替换为 64。接下来要解码的非终结符是 <ReplAlg>，其产生式规则 IV 有 3 种不同的选项。随后处理密码子值 71，考虑到 $71\ MOD\ 3 = 2$，选定的值为 r。类似地，<Assoc> 对应的下一个密码子 96 转换为 1，因为 $96\ MOD\ 8 = 0$。最后一个与指令缓存参数对应的非终结符 <PrefAlg>，由密码子值 123 确定；由于 $123\ MOD\ 3 = 0$，该值被替换为 m。接下来，映射过程继续处理密码子 210 和第二个 <CacheSizeB> 符号，其映射结果为 2048，对应数据缓存大小。然后，137 将 <LineSizeB> 映射为 16，基因 7 将 <ReplAlg> 映射为 f，而 5 将 <Assoc> 映射为 4。

此时，所有基因都已使用，但表达式仍未完全解码。不过，语法进化能够处理可变长度染色体；如果解码过程到达染色体末尾时仍有非终结符，语法进化就会回到第一个密码子，让解码继续完成。这个过程称为回绕。因此，在我们的示例中，此处发生了一次回绕。于是，<预取算法> 由第一个基因20解码得到 a。最后，解码将 <写策略> 映射为 n，对应的基因值为 35。

因此，我们的基因型示例得到了图4下半部分所示的表型，它表示一种缓存配置，其中指令缓存大小为 8192，指令缓存块大小为64，等等。

解码后，使用该表型调用缓存模拟器，以评估目标应用程序对应的配置，并根据式（4）中定义的适应度获取执行时间和能耗。在选定的代数结束时，语法进化会返回适应度值最高的缓存配置。

如图所示，语法既决定了要优化的具体参数，也决定了与缓存模拟器的通信方式，而缓存模拟器会提供适应度值。因此，更改语法即可采用不同的优化方法：固定某些参数的值、添加新参数、更改缓存模拟器、在缓存模拟器中使用可变长度参数（例如用于自定义算法）等。

这样，在参数的数量（及类型）固定且给定缓存模拟器的情况下，语法会迫使进化过程生成固定结构，而语法进化则负责优化该结构中的参数。因此，可以采用遗传算法、粒子群优化等定长算法来解决这个问题。然而，在这类方法中，参数的数量或类型发生变化，甚至模拟器发生变化时，都需要修改代码中的重要部分，主要是解码和适应度评估模块。因此，我们基于语法进化的方法具有很强的灵活性

同时也具备可扩展性，因为只需简单修改语法，就能纳入这些变化。

GE算法由我们的研究团队使用 Java 开发，并以 JECO库 [27] 的形式公开提供。

6 实验结果

为了测试我们的实验框架，我们从 Mediabench 基准测试套件中选取了十二个基准测试 [18]，这些基准测试能够代表多媒体应用。接下来，我们将介绍这些基准测试的特点。完整说明可在 Mediabench 网站上找到。

– JPEG 是一种用 C语言 编写的软件，可用于压缩（$cjpeg$）和解压缩（$djpeg$）全彩及灰度 JPEG 图像。– MPEG 是一种经过优化的实现，用于传输高质量数字视频。*MPEG* 编码器 基准测试 mark负责编码过程，而*MPEG* 解码器基准测试则负责解码过程。– GSM实现了一种可靠的语音识别算法。该算法基于欧洲GSM 06.10全速率语音转码临时标准的实现。$gsmenc$基准测试执行压缩帧处理，而 $gsmdec$ 执行解压缩处理。– $Pegwit$ 实现公钥加密和身份验证过程。该过程使用GF(2^{255})上的椭圆曲线、用于哈希处理的SHA1编码算法，以及对称分组密码 square。$Pegwitenc$ 和$pegwitdec$ 分别执行编码和解码。– EPIC（高效金字塔图像编码器）执行图像数据压缩（$epic$）和解压缩（$unepic$）。这些算法基于双正交临界采样二进小波分解，以及组合式游程长度／霍夫曼熵编码器。– ADPCM（自适应差分脉冲编码调制）实现语音压缩和解压缩算法。$Rawcaudio$ 执行压缩，而 $rawdaudio$ 执行解压缩。它们所使用的算法在线性 PCM采样（16位）上实现了4:1的压缩率。

我们的实验在一台配备英特尔 i5 660处理器（主频为 3.3 GHz）的机器上进行。由于这是一款 4 核处理器，我们最多同时运行 4 个实验。该机器配备 8 Gb 随机存取存储器，操作系统为 Ubuntu Desktop 14.04。

6.1 实验设置

如前所述，我们的实验框架根据 ARM架构进行参数化。随后，为了评估我们的方法，我们考虑大小范围为 512B 至 64KB、按 2 的幂递增的一级缓存。通过这组可选缓存大小，我们涵盖了 ARM9处理器系列中一些设备的多种缓存大小。根据这些特性，第 2 节描述了语法进化的搜索空间，其中结合了指令缓存和数据缓存。然而，在离线特性分析过程中，缓存特性分析针对各个存储器单独进行，且仅考虑硬件参数。在我们的搜索空间中，如图1所示，这些参数对应 8 种可能的缓存大小、8 种不同的相联度以及 4 种可能的块大小。因此，需要运行 256 次缓存特性分析。

使用我们的离线分析方法，我们利用 Trimaran 工具生成了 Mediabench 跟踪记录 [7]，并结合 SimpleScalar [5] 评估 ARM架构。基于上述内容，我们对 SimpleScalar 和 Trimaran 工具进行了适当修改，以生成能够由 Dinero IV 缓存模拟器处理的应用程序跟踪。

这类实验需要处理完整的应用程序跟踪，或执行一定数量的、能够反映完整程序行为的适当指令。这样可以避免捕获局部行为或阶段性行为。例如，阶段性行为会发生在应用程序运行开始时，此时必须将所有数据块从主存复制过来，因此对缓存存储器的所有访问都会未命中。因此，少量指令可能产生无法被视为具有代表性或结论性的结果。为此，我们对每个应用程序最多仿真 7.5×10^7 条指令；如果完整执行所需的指令总数低于该值，则仿真整个应用程序。这样做旨在在仿真时间、程序跟踪的大小以及适当的指令数量之间取得平衡。

此外，为了降低陷入局部最优解的概率，我们针对每个目标应用程序运行了 10 次语法进化。

表 2 展示了语法进化的参数。我们还采用了简单支配、单点交叉和整数翻转变异算子，其概率设置遵循 [10]中的建议。

如（4）中所述（见第 5.2 节），我们的算法需要基线缓存配置，以便计算个体的优劣。对于这些实验，我们选取了一种基线缓存配置，其与 GP2X

表 2 语法进化的参数。

代数	100
种群规模	50
交叉概率	0.9
变异概率	0.01

便携式游戏机的第一个核心的缓存相似；该游戏机采用双核 200 MHz ARM940T 处理器。

表3展示了基准配置中缓存的大小、块大小、路数（相联度）、替换策略、搜索算法，以及数据缓存的分配策略。

基准缓存的执行时间和能耗按照第4节所述的模型计算。

6.2 目标函数

首先，我们分析目标函数值方面的结果。需要再次说明，目标值1对应于基准缓存配置运行每个基准测试时的执行时间和能耗。因此，我们通过最小化目标函数来降低这两项指标。

表4展示了各基准测试的以下统计数据：目标值、相对于所找到的最佳目标值的偏差百分比，以及取得最佳结果的次数。

如表所示，平均目标值介于0.47和0.30之间，这意味着改进幅度为53%至70%，目标值平均改进62%。因此，优化取得了相当显著的目标值改进。此外，标准差较低，表明GE在各次优化运行中取得的结果非常接近。事实上，在12个基准测试中，有7个的最佳结果出现频率不低于70%。具体而言，GE在$pegwitdec$、$pegwitenc$、$rawcaudio$ 和$rawdaudio$ 基准测试中取得了出色的结果。另一方面，尽管GE在$djpeg$ 和$unepic$中仅各取得一次最佳值，但偏差分别低于3.1%和0.7%，这同样是非常好的结果。

这些结果证明，即使代数和种群规模较小，像 GE 这样的元启发式算法也能够为缓存优化问题找到解决方案，使目标函数降低超过 62%。

6.3 优化运行时间

我们还对优化运行时间（RT）进行了另一项分析。表5列出了平均优化运行时间（小时）、GE 的平均评估次数占预期最大评估次数的比例、评估一种缓存配置的平均运行时间（秒）、若执行全部评估时 GE 的最长优化运行时间，以及实验相对于该运行时间节省的时间。

在运行优化的同时，我们测量了评估每种缓存配置所花费的时间。如表中所示，在某些基准测试中，平均评估时间较长，例如 $djpeg$、$mpegdec$ 和 $mpegenc$。考虑到这些时间值，我们计算了在执行 GE 时运行 5000 次评估所需的优化时间，也就是说，假如该算法不为已评估个体实现记忆功能时所需的时间。

从节省比例列可以看出，执行时间减少了 93.6% 以上。因此，显然，由于能够记住已评估个体，我们的 GE实现可以在合理的时间内运行。

优化运行时间缩短有两个原因。一方面，评估次数较少，因为 GE算法的探索会到达最小值区域，使种群收敛到该最小值。换言之，种群中的个体在代际演进过程中会变得越来越相似。另一方面，由于我们的算法会记住每个个体的评估结果，因此不会将先前评估过的配置发送到外部模拟器，而是从 GE 的记忆中读取。这一过程非常快速，因为我们使用哈希映射实现了这一功能。

6.4 执行时间和能耗

如第 5.2 节所述，目标函数同时纳入了给定缓存配置的执行时间和能耗值。随后，在获得优化后的缓存配置后，我们将展示如何从执行时间和能耗的角度解读这些结果。

图 5 绘出了每个基准测试运行时的平均能耗，其中基准缓存的能耗设为 100%。平均而言，我们获得的解决方案达到了基准缓存能耗的 3.04%。

图 6 绘出了各基准测试运行的平均执行时间，其中基准缓存的能耗为 100%。

表3 基准缓存配置由指令缓存和数据缓存的参数构成。

指令缓存				
Size	**块大小**	**相联度**	**替换策略**	**搜索算法**
16 KB	32 B	4	LRU	按需

数据缓存					
Size	**块大小**	**相联度**	**替换策略**	**搜索算法**	**分配策略**
16 KB	32 B	4	LRU	按需	回写

表4 目标函数值。

基准测试	平均目标	偏差（%）	最佳数量
cjpeg	0.3986650026	0.74%	8
djpeg	0.4746031647	3.06%	1
epic	0.3365548961	1.24%	7
unepic	0.3977187056	0.65%	1
gsmdec	0.3857290316	1.15%	7
gsmenc	0.4442631856	1.82%	4
MPEG 解码器	0.3682433302	1.09%	6
MPEG 编码器	0.3631493163	0.89%	3
Pegwit 解码器	0.3663486721	0.01%	9
Pegwit 编码器	0.3568943181	0.00%	10
rawcaudio	0.3052829963	0.00%	8
rawdaudio	0.31533024	0.56%	9

表5 运行时间数值。对于每个基准测试，表中显示平均运行时间（小时）、平均评估次数与最大语法进化评估次数的对比、一次缓存评估的平均运行时间（秒）、语法进化的最大运行时间（不进行评估召回），以及相对于语法进化最大运行时间的运行时间节省。

基准测试	运行时间（小时）	评估次数（%）	评估时间（秒）	最长语法进化时间（小时）	运行时间节省
cjpeg	2.185	5.96%	26.200	36.389	93.99%
djpeg	2.647	5.74%	33.064	45.923	94.24%
epic	1.405	5.87%	17.148	23.816	94.10%
unepic	0.350	5.79%	4.340	6.028	94.19%
gsmdec	0.004	6.33%	0.043	0.060	93.67%
gsmenc	0.003	5.58%	0.041	0.057	94.41%
MPEG 解码器	2.275	5.69%	28.805	40.006	94.31%
MPEG 编码器	2.031	5.80%	25.209	35.013	94.20%
Pegwit 解码器	0.945	6.38%	10.667	14.815	93.62%
Pegwit 编码器	0.869	5.98%	10.466	14.537	94.02%
rawcaudio	0.372	5.95%	4.483	6.227	94.03%
rawdaudio	0.274	5.89%	3.343	4.642	94.10%

平均而言，我们获得的解决方案达到了基准的 24.74%。

简而言之，相较于商用基准缓存，我们将缓存行为在能耗方面提升了 97%，同时将执行时间缩短了 75%。与预期中无评估召回率的语法进化所需的时间相比，这些结果是在极短的运行时间内取得的。

为清楚起见，我们想说明，基准配置是语法进化流程进行优化时所依据的参考点。事实上，由于我们的框架使用缓存模拟器（本例中为 Cacti）的特征数据，只要向框架提供相应的基准特征值，就可以优化任何基准配置。

此外，鉴于基准缓存的基准测试结果尚未优化，

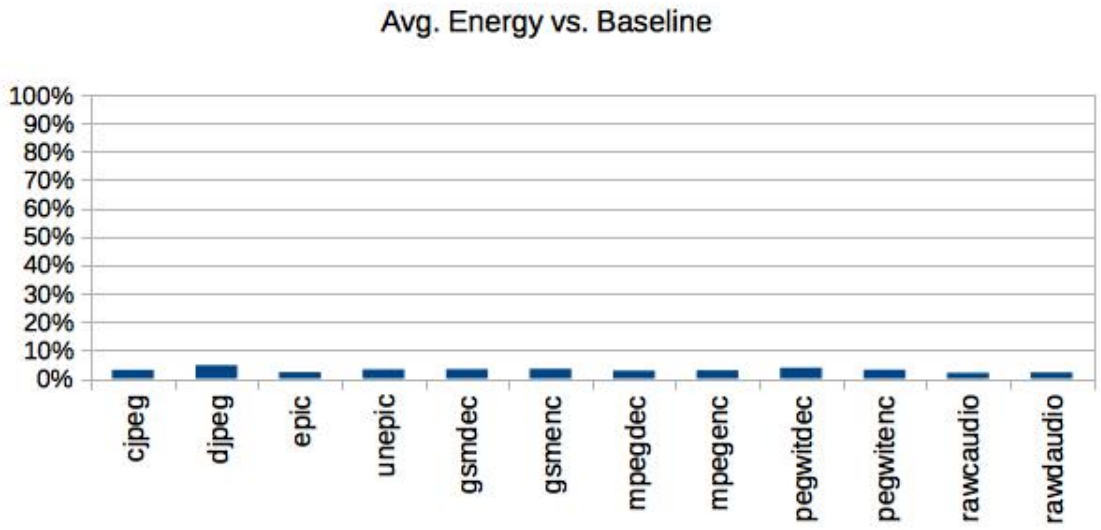

图5 平均　　各基准测试中所得解相对于基准（100%）的平均能耗　　.

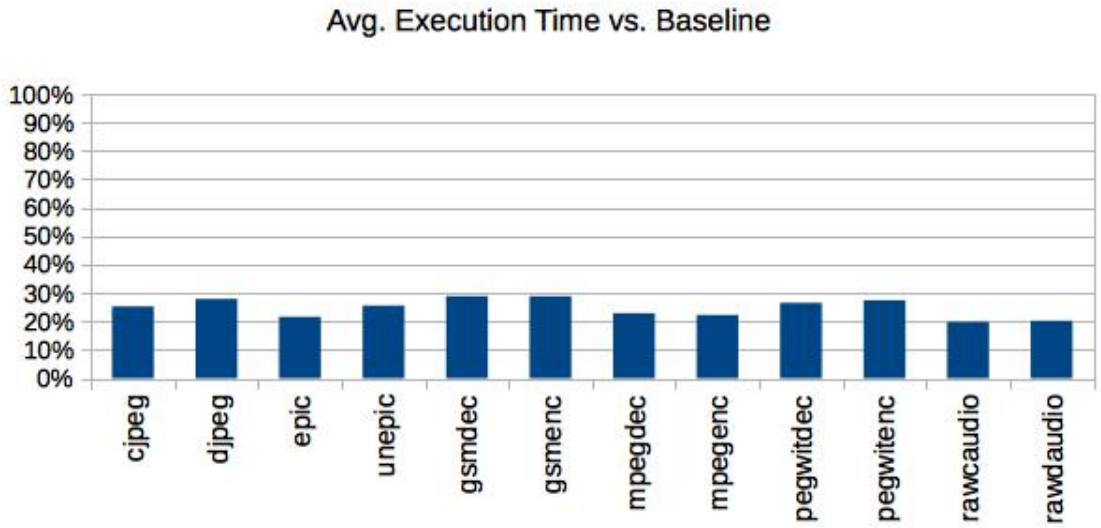

图6 各基准测试中所得解相对于基准（100%）的平均执行时间。

因此，GE（经过优化）所取得的改进应视为潜在性能提升的参考。

7 结论与未来工作

平板电脑、智能手机、GPS导航系统等嵌入式设备中的多媒体应用显著增加，促使研究人员致力于改善执行时间、降低能耗，或同时实现这两项目标。为了缩短执行时间，当前的嵌入式系统采用了类似于台式计算机的缓存层次结构。然而，各种各样的缓存配置使搜索最佳缓存配置变得复杂。更不用说，应用程序的内存访问模式会直接影响执行时间和能耗。因此，多年来，人们提出了新的高效技术，用于探索并确定适用于不同应用程序的最佳缓存配置。然而，我们尚未在文献中发现任何研究能够同时优化给定应用程序集合中指令和缓存存储器的所有配置参数。

本文介绍了一种基于静态分析和元启发式算法的优化框架，

用于针对将在目标嵌入式系统上运行的一组应用程序，自动搜索最佳缓存配置。我们的优化方案考虑了 11 个不同参数，用于配置指令缓存和数据缓存。该框架分为两个阶段。第一阶段为离线阶段，负责提供程序跟踪以及缓存存储器可选硬件配置的特征。第二阶段是优化过程，运用语法进化，得出能使各目标应用程序获得最佳表现的缓存配置。为了评估缓存配置，语法进化借助 Dinero IV 缓存模拟器获取各配置的性能。这样一来，语法进化的语法便于与 Dinero IV 通信，因为表型由模拟器调用所需的参数和格式构成。

此外，我们的方法只需修改相应的语法，即可修改待优化参数的数量，以及更换缓存模拟器。此外，如果缓存模拟器支持可变长度参数，适当的语法便能够生成正确的模拟器调用，而无需对我们的优化框架进行任何修改。

我们以实际的缓存配置作为基准，使用来自 Mediabench 基准测试的 12 个应用程序对所提出的方法进行了测试。与基准缓存相比，我们的结果显示，能耗平均降低了 97%，执行时间平均缩短了 75%。

我们未来的计划包括努力简化流程，并设计一个有效的框架，使其能够针对一组目标应用程序同时定制优化过程。此外，我们将考虑内存操作在目标应用程序中所代表的不同负载，以便在优化过程中更精确地平衡其权重。事实上，这项研究之后自然的下一步将是分析执行时间与能耗结果之间的相互影响，并考虑多目标优化。

参考文献

1. http://www.grammatical-evolution.org/pubs.html (2015) 2. ARM Ltd.：ARM9处理器系列。http://www.arm.com/products/processors/classic/arm9/ (2013) 3. ARM946E-S TM：技术参考手册。http://www.arm.com/products/processors/classic/arm9/ arm946.php (2014) 4. Brabazon, A., O'Neill, M.：《用于金融建模的生物启发式算法》。施普林格 (2006) 5. Burger, D., T.M.Austin：《SimpleScalar工具集》，版本 2.0。载于：技术报告，第 CS–TR–97–1342 页。威斯康星大学麦迪逊分校 (1997)

6. Byrne, J., McDermott, J., Galva´on-Lo´opez, E., O'Neill, M.：为交互式进化三维设计实现直观变异算子。载于：《进化计算》（CEC），2010年IEEE进化计算大会，第1–7页（2010）。DOI 10 5586485 7 1109 2010 . /CEC. . . Chakrapani, L.N., Gyllenhaal, J., mei W. Hwu, W., Mahlke4, S.A., Palem1, K.V., Rabbah5, R.M.：Trimaran：一个用于指令级并行研究的基础设施。载于：指令级并行。《计算机科学讲义》（2005）。URL http://www.trimaran.org/ 8 。Chen, L., Zou, X., Lei, J., Liu, Z.：面向低功耗嵌入式系统的动态可重配置缓存。载于：自然计算，2007年。第三届国际自然计算会议（ICNC 2007），第5卷，第180–184页（2007）。DOI 10 346 9 1109 2007 . /ICNC. . . Dani, A., Srikant, Y., Amrutur, B.：面向平铺式芯片多处理器的高效缓存探索方法。载于：高性能计算（HiPC），2012年第19届国际会议，第1–6页（2012）。DOI 10.1109/HiPC.2012. 6507524 10. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.：一种快速且精英式的多目标遗传算法：NSGA-II。《IEEE进化计算汇刊》6(2)，182–197 996017 11 2002 10 1109 4235 ( )。DOI . / . . Dempsey, I., O'Neill, M., Brabazon, A.：《动态环境下语法进化基础》，*Studies in Computational Intelligence*，第194卷。施普林格（2009）。URL http://www.springer.com/engineering/book/978-3-642-00313-412. Edler, J., Hill, M.D.：Dinero IV，基于跟踪驱动的单处理器缓存模拟器（1998）。URL http://pages.cs.wisc.edu/~markhill/DineroIV/13. Galv´an-L´opez, E., Swafford, J., O'Neill, M., Brabazon, A.：使用语法进化来演化 Ms. Pac-Man 控制器。载于：C. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekrt, A. Esparcia-Alcazar, C.K. Goh, J. Merelo, F. Neri, M. Preu, J. Togelius, G. Yannakakis（编），《进化计算应用》，*Lecture Notes in Computer Science* 6024, 161 ，第...–170页。施普林格·柏林·海德堡（2010）。DOI 10.1007/978-3-642-12239-2\ 17。URL http://dx.doi.org/10.1007/978-3-642-12239-2_1714. Gordon-Ross, A., Lau, J., Calder, B.：面向高度可配置两级缓存层次结构的基于阶段的缓存重配置。载于：第18届ACM五大湖超大规模集成电路研讨会论文集，GLSVLSI '08，第379–382页。ACM，美国纽约（2008）。DOI 10.1145/1366110. 1366200。URL http://doi.acm.org/10.1145/1366110.1366200 15. Hennessy, J.L., Patterson, D.A.：《计算机体系结构：定量方法》。摩根·考夫曼（2011）16. Janapsatya, A., Ignjatovic, A., Parameswaran, S.：为嵌入式系统寻找最优L1缓存配置。载于：设计自动化，2006年亚洲及南太平洋会议，第6页（2006）。DOI 10.1109/ASPDAC. 2006.1594783 17. Koza, J.R., Poli, R.：优化、搜索与决策支持导论教程（2003）18. Lee, C., Potkonjak, M., Mangione-Smith, W.H.：Mediabench：用于评估和综合多媒体与通信系统的工具。载于：第30届ACM/IEEE国际微体系结构研讨会论文集，MICRO 30，第330–335页。IEEE计算机学会，美国华盛顿特区（1997）。URL http://dl.acm.org/citation.cfm?id=266800.266832

19. Mamidipaka, M., Dutt, N.：eCACTI：一种用于片上缓存的增强型功耗估算模型。技术报告 TR-04-28，CECS，加州大学欧文分校（2004）20. Naz, A., Kavi, K., Rezaei, M., Li, W.：论分离式数据缓存对嵌入式应用的价值。《SIGARCH计算机体系结构新闻》**34**(1)，19–26（2005）。DOI 10.1145/1147349.1147355。URL http://doi.acm.org/10.1145/1147349.114735521. O'Neill, M., Hemberg, E., Gilligan, C., Bartley, E., McDermott, J., Brabazon, A.：GEVA——Java语法进化。《SIGEVOlution》**3**(2)，17–22（2008）。DOI http://doi.acm.org/10 1527066 22 1145 1527063 . / . . O'Neill, M., Ryan, C.：自动生成缓存算法。载于：《工程与计算机科学中的进化算法》，第127–134页。John Wiley & Sons（1999）23. O'Neill, M., Ryan, C.：语法进化。《IEEE进化计算汇刊》**5**(4)，349–358（2001）24. O'Neill, M., Ryan, C.：《语法进化：任意语言中的进化式自动编程》。Kluwer Academic Publishers（2003）25. Palesi, M., Givargis, T.：使用遗传算法进行多目标设计空间探索。载于：硬件/软件协同设计，2002年。第十届国际研讨会论文集，第67–72页（2002）。DOI 10 1003603 26 1109 2002 . /CODES. . . Perez, D., Nicolau, M., O'Neill, M., Brabazon, A.：使用语法进化为马里奥人工智能竞赛演化行为树。载于：C. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekrt, A. Esparcia-Alc´azar, J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, G. Yannakakis（编），《进化计算应用》，*Lecture Notes in Computer Science*，第6624卷，第123–132页。施普林格·柏林·海德堡（2011）。DOI 10.1007/978-3-642-20525-5_13。URI http://dx.doi.org/10.1007/978-3-642-20525-5_1327 。Risco-Martín, J.L., Colmenar, J.M.：Java进化计算库（JECO）。可访问：https://sourceforge.net/projects/jeco（2013）28. Risco-Mart´ın, J.L., Colmenar, J.M., Hidalgo, J.I., Lanchares, J., D´ıaz, J.：一种应用语法进化自动优化动态内存管理器的方法。《系统与软件杂志》**91**(0)，109–123（2014）。DOI http://dx.doi.org/10.1016/j.jss.2013. 12.044。URL http://www.sciencedirect.com/science/article/pii/S0164121214000171x29 。Shaker, N., Nicolau, M., Yannakakis, G., Togelius, J., O'Neill, M.：使用语法进化为《超级马里奥兄弟》演化关卡。载于：计算智能与游戏（CIG），2012年IEEE会议，第304–311页 6374170 30 2012 10 1109 2012 ( )。DOI . /CIG. . . Silva-Filho, A., Bastos-Filho, C., Falcao, D., Cordeiro, F., Castro, R.：一种面向两级缓存层次结构的优化机制，旨在使用NSGA-II算法提升能耗和性能。载于：计算机体系结构与高性能计算，2008年。第20届国际研讨会（SBAC-PAD '08），第19–26页（2008 9 31 10 1109 2008）。DOI . /SBAC-PAD. . . Varma, A., Debes, E., Kozintsev, I., Jacob, B.：英特尔XScale嵌入式微处理器的指令级功耗。载于：SPIE第17届电子成像科学与技术年度研讨会（2005）32. Wang, W., Mishra, P.：软实时嵌入式系统中两级缓存的动态重配置。载于：超大规模集成电路，2009年。IEEE计算机学会国际研讨会（ISVLSI '09）年度研讨会，页码 145–150（2009）。DOI 10.1109/ISVLSI.2009.22 33. Wang, W., Mishra, P., Gordon-Ross, A.：软实时系统的动态缓存重配置。ACM嵌入式计算系统汇刊**11**(2)，28:1–28:31（2012）。DOI 10.1145/2220336.2220340。URL http://doi.acm.org/10.1145/2220336.222034034. Wang, W., Mishra, P., Ranka, S.：实时多核系统的动态缓存重配置与分区，以实现能耗优化。载于：第48届设计自动化会议论文集，页码 948–953。ACM，美国纽约（2011）。DOI 10.1145/2024724.2024935。URL http://doi.acm.org/10.1145/2024724.2024935