遗传算法和关键事件禁忌搜索相融合的ARM/Thumb处理器指令选择

吴圣宁 李思昆
(国防科学技术大学计算机学院 长沙 410073)

摘 要 面向嵌入式系统的编译器,往往需要同时考虑目标代码的性能、大小和功耗等相互冲突的目标. ARM双指令集处理器,在具备通常的32位ARM指令集基础上,还支持一个缩减的16位Thumb指令集,因而为代码优化提供了多个目标之间折衷的机会. 由于同一个程序的Thumb代码比相应的ARM代码执行更多的指令,因此虽然前者常比后者占用更少的存储空间,但消耗更多的运行时间. 针对这种现象,文中建议一个混合演化算法,通过把程序的不同部分有选择地编译成ARM或Thumb指令集代码,使得可灵活地权衡目标代码大小和运行时间. 文中的方法基于遗传算法和关键事件禁忌搜索相融合,后者用来局部搜索. 指令选择以函数为单位,从对程序动态行为的profiling分析求得程序运行时间. 实验结果表明,文中的技术可有效地、灵活地权衡目标代码大小和性能,并且适用于其它的双指令集处理器.

关键词 演化算法;指令选择;ARM/Thumb
中图法分类号 TP314

Instruction Selection for ARM/Thumb Processors Based on a Genetic Algorithm Coupled with Critical Event Tabu Search

WU Sheng-Ning LI Si-Kun
(*School of Computer Science, National University of Defense Technology, Changsha* 410073)

Abstract In the embedded domain, not only performance, but also memory and energy are important concerns. A dual instruction set ARM processor, which supports a reduced Thumb instruction set with a smaller instruction length in addition to a full instruction set, provides an opportunity for a flexible tradeoff between these requirements. For a given program, typically the Thumb code is smaller than the ARM code, but slower than the latter, because a program compiled into the Thumb instruction set executes a larger number of instructions than the same program compiled into the ARM instruction set. Motivated by this observation, this paper proposes a hybrid evolutionary algorithm that can be used to enable a flexible tradeoff between the code size and execution time of a program by using the two instruction sets selectively for different parts of a program. The proposed approach is based on a genetic algorithm coupled with critical event Tabu Search as the local search. The instruction set is determined to be used for each function, and the execution time is got based on the profiling analyses of the dynamic behavior of a program. The experimental results show that the proposed technique can be effectively used to make the tradeoff between a program's code size and execution time and can provide much flexibility in code generation for dual instruction set processors in general.

Keywords evolutionary optimization; instruction selection; ARM and Thumb

收稿日期:2005-11-06;修改稿收到日期:2007-01-07. 本课题得到国家自然科学基金(90207019)资助.**吴圣宁**,1971年生,博士研究生,主要研究方向为嵌入式系统编译器、操作系统、人工智能. E-mail: kanvard@yahoo.com.cn. **李思昆**,男,1941年生,教授,博士生导师,研究领域为电子CAD、系统芯片设计方法学、虚拟现实.

1 引　言

嵌入式系统常有多个相互冲突的要求，例如性能高、体积小、功耗低、价格低等. 因此，面向嵌入式处理器的编译器需要同时针对多个目标优化目标代码.

双指令集处理器对编译器生成代码提出了挑战，同时也为灵活地权衡程序的代码大小和运行时间提供了机会. ARM 处理器[1]在嵌入式领域得到了广泛应用. 除了 32 位 ARM 指令集，这些处理器也支持 16 位 Thumb 缩减指令集. 一条 Thumb 指令比相应的 ARM 指令占用存储空间小，但后者能比前者执行更多的操作. 因此，Thumb 指令集是以代码性能的损失来换取更小的存储空间.

本文的目标是，当为 ARM/Thumb 处理器生成代码时，合理利用上述特征，以灵活地权衡代码大小和性能. 目前对双指令集处理器代码生成的研究，一般利用简单启发式算法，生成代码的质量不能满足嵌入式系统的严格要求.

许多编译优化问题是 NP-hard 问题. 由于要求具有较快的编译时间，编译优化技术典型地局限于可较快执行的算法(小于 $O(n^2)$). 因此，很多编译优化问题常利用简单的线性时间的启发式算法求解. 对嵌入式系统来说，较快的编译时间没有代码的质量重要. 尽管这些启发式算法执行速度快，但生成代码的质量通常有较大的提高余地. 嵌入式系统要求在任何合理的时间内生成尽可能占用存储空间小同时又性能高的代码.

NP-hard 代码优化问题是计算上难解的. 实践中，我们常满足于接近最优解的“较好”解. 在过去二十年中，对组合优化问题，出现了一种新的在较短计算时间内给出高质量解的近似算法. 这些方法现在统称为元启发式(meta-heuristics)算法. 除了单个解搜索算法(例如模拟退火[2]、禁忌搜索[3])，基于群体的元启发式算法得到越来越广泛的重视(例如遗传算法[4]、蚁群优化算法[5]等). 这些基于群体的算法能够在一次模拟过程中发现多个优化解，因而特别适合于求解多目标优化问题.

对双指令集处理器，同时优化程序的代码大小和运行时间是一个复杂的优化问题. 为克服简单启发式算法代码生成的质量问题，本文提出一个遗传算法和关键事件禁忌搜索相融合的算法，通过使用完全和缩减两个指令集选择性地编译各子程序，来灵活地权衡程序的代码大小和运行时间. 指令选择以函数为单位，基于对程序动态行为的 profiling 分析估算程序总体运行时间.

本文第 2 节介绍相关工作；第 3 节对提出的问题给出形式化描述；第 4 节详细描述针对 ARM/Thumb 处理器的指令选择混合演化算法；第 5 节讨论实验结果；第 6 节总结并展望将来的工作.

2 相关工作

Halambi 等[6]提出了一个生成完全、缩减混合指令集代码的方法，可有效减少代码的存储空间. 此技术把连续的指令编成可编译为缩减指令的组，并基于对结果代码大小的估算，来决定是否要确实编译为缩减指令.

Krishnaswamy 和 Gupta[7]建议了基于函数的粗粒度的几个启发式方法，重点在于提高针对运行时间和功耗的指令 Cache 性能. 他们还提出了一个细粒度方法，由于 ARM 和 Thumb 模式切换指令的开销，其性能仅和粗粒度方法相当.

Lee 等[8]通过基于路径的收益分析技术，提出了针对程序基本块的指令选择方法，目的是在满足代码大小的约束条件下，使得程序运行时间最小.

总之，上述这些算法都利用简单启发式方法来选择子程序的指令集类型，不能很好地满足嵌入式系统目标代码生成的严格要求.

3 问题描述

这一节给出所要解决问题的形式化描述. 一个程序可以表示为函数调用图 $G=\langle V,E\rangle$，其中 V 是函数的结合，E 是 V 中元素有序偶的集合，称为边，表示函数调用关系. 假定程序有 n 个函数，$V=\{v_i \mid i=1,2,\cdots,n\}$，$E=\{e_{ij}=\langle v_i,v_j\rangle \mid v_i$ 调用 $v_j\}$. 设 $s_{v_i}^{\mathrm{ARM}}$，$s_{v_i}^{\mathrm{Thumb}}$ 分别为函数 v_i 编译成 ARM，Thumb 模式时的代码大小；设 $t_{v_i}^{\mathrm{ARM}}$，$t_{v_i}^{\mathrm{Thumb}}$ 分别为函数 v_i 编译成 ARM，Thumb 模式时的运行时间.

为每个函数 v_i 附加一个决策变量 x_i，

$$x_i=\begin{cases}1, & v_i \text{ 编译成 ARM 指令级}\\ 0, & v_i \text{ 编译成 Thumb 指令级}\end{cases} \tag{1}$$

下面估算程序的代码大小和运行时间. 首先，计算代码大小. 假设 $overhead_i^s$ 为 v_i 和其所有父节点之间模式切换指令的存储开销. 程序总的代码大小可

由每个函数大小和其模式切换开销累加得到

$$Size_{\text{program}}=\sum_i (s_{v_i}^{\text{ARM}}\cdot x_i+s_{v_i}^{\text{Thumb}}\cdot(1-x_i)+overhead_i^s) \quad (2)$$

$$overhead_i^s=c_s\cdot(\bigcup_j((x_j+x_i)\bmod 2)),\ \exists e_{ji}\in E \quad (3)$$

其中,c_s表示一次模式切换的指令存储开销.

然后,计算程序总的运行时间.假定函数v_i的父节点的个数为k,父节点为$v_{i_1},v_{i_2},\cdots,v_{i_k}$,$v_i$被父节点调用的次数分别为$c_{i_1},c_{i_2},\cdots,c_{i_k}$,和父节点关联的决策变量分别为$x_{i_1},x_{i_2},x_{i_k}$.程序总的运行时间可由每个函数运行时间和其相关的模式切换时间开销累加得到

$$Time_{\text{program}}=\sum_i\Big(\sum_{j=1}^{k}((t_{v_i}^{\text{ARM}}\cdot x_i+t_{v_i}^{\text{Thumb}}(1-x_i)+overhead_{i_j}^t)\cdot c_{i_j})\Big) \quad (4)$$

$$overhead_{i_j}^t=c_t\cdot((x_{i_j}+x_i)\bmod 2) \quad (5)$$

其中,$overhead_{i_j}^t$是父节点v_{i_j}和v_i之间模式切换指令的时间开销,c_t表示一次模式切换指令的时间开销.

总之,本文解决的问题可表示如下:

$$\text{Minimize } Time_{\text{program}} \quad (6)$$

$$Size_{\text{program}}\le MaxSize$$

更具体地表示为

$$\text{Minimize } \sum c_j\cdot x_j+c_0 \quad (7)$$

$$\text{subject to } \sum s_j\cdot x_j\le s_0,\ x_j \text{ binary}$$

如果考虑模式切换开销,问题约束和目标函数的系数需要动态计算,这是因为对某个函数指令类型的指派也影响到和其有调用关系的其它函数指令类型的指派.

4 双指令集处理器代码选择的混合演化算法

本文的混合演化算法基于 Deb 的遗传算法 NSGA-Ⅱ(Elitist Non-Dominated Sorting Genetic Algorithm)[9]、Glover 和 Kochenberger 的关键事件禁忌搜索 CETS(Critical Event Tabu Search)[10].本文利用前者作为全局控制,对后者进行了修改作为局部搜索子过程.关键事件禁忌搜索在遗传算法每一次迭代的最后阶段使用,基于演化过程的结果进行解空间局部搜索.

4.1 NSGA-Ⅱ

NSGA-Ⅱ是当前性能最好的遗传算法之一.其原理简述如下.首先,由大小为N的父群体P_t生成子群体Q_t,把这两个群体合在一起形成大小为$2N$的群体R_t.对R_t的个体进行非支配(non-dominated)排序,把R_t的个体划分为若干个等级,每个等级中的个体相互不能支配,但高等级中的个体可支配低等级中的个体.其次,形成大小为N的子群体.按照等级由高到低的顺序,用R_t各等级中的个体填充新子群体的N个位置,每次选择一个等级的所有个体.最终,当子群体中的剩余位置不够安置下一个等级时,选择此等级中的部分个体填满子群体的剩余位置,这些个体代表所在等级最不拥挤的解空间.NSGA-Ⅱ解的多样性通过解之间的拥挤比较过程来实现,不需要额外的小生境参数.

4.2 关键事件禁忌搜索

下面首先简要介绍关键事件禁忌搜索 CETS 的原理,然后详细说明本文的混合算法对它进行的修改.

CETS 围绕着关键事件来组织核心记忆功能.它利用一个策略振荡机制在可行和不可行解空间之间的界限两侧,进行系统的可变深度的探测,以实现 intensification 和 diversification 之间的平衡;并利用代理约束(surrogate constraint)分析来派生决策变量的选择规则.

CETS 在求解过程中,在构造和破坏两个阶段间不断切换.构造阶段中,连续地把决策变量设置为 1,而在破坏阶段中,连续地把决策变量设置为 0.关键事件指的是构造阶段中在将要进入不可行解空间的最后时刻得到解,和破坏阶段中第一次重新获得可行解.

搜索的过程即为在解空间中移动(move)的过程.一次移动指的是一个决策变量的跳转.一个解的邻域为在当前状态下所有可能的移动的集合;邻域的大小为决策变量的个数.移动的评价基于代理约束和禁忌惩罚.参数 span 决定了振荡的幅度,影响可行域边界两侧的振荡.

由于本文仅使用 CETS 作为局部搜索,因此所建议的方法和文献[12]有以下主要区别:

(1)本文仅使用短期记忆信息;

(2)因为只有一个程序代码大小的约束,所以对代理约束的计算进行了修改.

4.2.1 决策变量选择规则

在策略性振荡的构造阶段,选择下一个设置为 1

的变量，使得

$$\text{Minimize}\ (c_j/s_j : x_j=0) \tag{8}$$

对称地，在破坏阶段，选择下一个设置为 0 的变量，使得

$$\text{Maximize}\ (c_j/s_j : x_j=1) \tag{9}$$

其中，如第 3 节的公式(7)所示，c_j 和 s_j 分别为决策变量在目标函数和存储空间约束中的系数. 这些规则被禁忌约束和惩罚因子修改后，选择下一个跳转的决策变量. 在构造阶段，优先选择这样的函数，即模式切换时消耗单位预算存储空间能使得程序的运行时间减少最多；在破坏阶段，优先选择这样的函数，即模式切换时消耗单位预算存储空间能使得程序的运行时间减少最少.

4.2.2　禁忌惩罚的计算

为搜索更广阔的解空间，需要对导致返回局部最优解的移动给予惩罚. 下面计算惩罚因子. 假设 $ratio(j)$ 表示变量 x_j 的目标函数和存储空间约束的对应系数的比值，$count$ 表示自最近一次构造和破坏阶段转换后已经选择的变量的个数，j^* 为下一个将要被选择的变量. 在构造阶段，我们按照如下方法选择 j^*(需满足 $x_j=0$)：

如果 $count>k$，则

$$value(j)=ratio(j) \tag{10}$$

如果 $count\leq k$，则

$value(j)=ratio(j)-PEN\_R\cdot \boldsymbol{TABU\_R}(j)$，

其中，$\boldsymbol{TABU\_R}$ 是短期禁忌向量，表示最近的 t(禁忌向量长度)个解被禁忌，PEN_R 为禁忌惩罚权值. 这时，j^* 即为 $value(j)$ 最小的变量. 类似地，在破坏阶段，j^*(需满足 $x_j=1$)即为 $value(j)$ 最大的变量. 参数 k 用来调节解空间搜索的发散度.

5　实验结果和讨论

本节给出针对双指令集处理器 ARM7TDMI 的算法实验结果，并和一个贪婪简单启发式算法相比较.

我们扩展 MachineSUIF 编译器[11] 的 HALT (the Harvard Atom-Like Tool)，作为程序的 profiling 工具，由此获得每个函数的执行次数. 函数 ARM、Thumb 模式的代码大小由此函数两种模式下的指令序列估算得到. 函数运行时间由此函数的指令条数估算得到近似值. 如果考虑 Cache，则 Thumb 模式的 Cache 命中率和取指令效率比 ARM 的高. 为简单起见，我们没有考虑 Cache 的影响，虽然这增加了和实际情况的误差，但并不影响问题的本质.

为验证算法的有效性，我们实现了所提出的技术并用一组 benchmark 程序做实验. SNU-RT benchmarks① 是面向实时嵌入式系统的一组应用程序，没有库函数调用，我们从中选取 4 个函数较多的程序：$fft1$，$fft1k$，fir，lms.

为了和贪婪启发式算法相比较，我们应用所建议的算法，为每个 benchmark 程序生成不同版本的代码. 假定某个程序有 $f_1, f_2, \cdots, f_n$ 这 n 个函数，$S_{f_i}^{\text{ARM}}$ 和 $S_{f_i}^{\text{Thumb}}$ 分别表示函数 f_i 编译成 ARM、Thumb 模式时的代码大小，则我们可以得到此程序总的代码大小的最小值和最大值：$MinTotal=\sum_i \min\{S_{f_i}^{\text{ARM}}, S_{f_i}^{\text{Thumb}}\}$，$MaxTotal=\sum_i \max\{S_{f_i}^{\text{ARM}}, S_{f_i}^{\text{Thumb}}\}$. 代码大小的总预算为 $\Delta size=MaxTotal-MinTotal$. 为生成不同版本的代码，我们设置代码大小预算分别为 0，$\Delta size$ 的 10%，20%，…，100%，然后和 $MinTotal$ 相加得到不同的代码大小约束.

对每个应用程序，算法运行 1000 次迭代后终止. 在所有的实验中，遗传算法参数设置如下：population $size=100$，probability of crossover of binary $variable=0.9$，probability of mutation of binary $variable=0.5$，$seed=0.5$；禁忌搜索参数设置如下：禁忌向量长度 $t=3$，惩罚因子参数 $k=4$，振荡参数 $p1=3$，$p2=5$. 参数的设置参照了求解本文以及其它组合优化问题中的经验. 实验结果验证了在程序的代码大小和性能之间存在明显的折衷问题，当增加存储空间时，代码运行时间逐渐减小，且这种现象随不同的 benchmark 而不同.

对于所有的 benchmark 程序，即使增加较小的存储空间，也能使得运行时间显著减少. 这是因为所有程序中的 $fabs$ 函数执行得最频繁，且在存储空间预算增加的早期阶段就被选择转换为 ARM 代码.

关于 $fft1$ 程序，如果基于贪婪启发式算法，则当代码大小的预算达到 $\Delta size$ 的 10% 时(相当于空间约束为 8458.3 字节)，$fabs$ 函数可被转换为 ARM 代码. 当空间预算达到 $\Delta size$ 的 60% 时(相当于空间约束为 8759.8 字节)，sin 函数被加入转换为 ARM 模式的函数集合. 当空间预算为 $\Delta size\times 90\%$(相当于空间约束为 8940.7 字节)，$fft1$ 函数被选择. 当达到总的空间预算 $\Delta size$ 时，所有的函数都能

① SNU Real-Time Benchmark Suite. http://archi.snu.ac.kr/realtime/benchmark

转换为 ARM 代码. 相对于贪婪启发式算法,本文的算法能更好地权衡代码大小和运行时间. 例如,当空间预算达到 $\Delta size \times 10\%$ 时,除了 *fabs* 函数,log 函数也可加入 ARM 代码集合.

fft1k 程序代码大小和运行时间的折衷类似于 *fft1* 程序,但前者运行时间的下降在开始阶段并不明显,这是因为在 *fabs* 函数之前有其它的函数首先被选择转换为 ARM 模式.

对于 *lms* 程序,由于随着存储空间的增加,其函数可渐进地加入 ARM 代码集合,因此程序的运行时间随着代码存储空间的增加而平滑地减小.

和 *lms* 程序类似,benchmark 程序 *fir* 的运行时间随着存储空间的增加近似线性地减少. 不同的是当存储空间预算达到 $\Delta size \times 50\%$ 时,除了 *main* 函数,其它的函数都已经加入 ARM 代码集合,因此程序的运行时间随着空间的增加没有明显的减小.

给定不同的存储空间预算(0,$\Delta size$ 的 10%,20%,…,100%),我们把建议的算法和一个简单的贪婪启发式算法相比较,此算法总是优先选择执行频率高的函数. 如图 1 所示,对所有的 benchmark 程序,给定相同的代码大小约束,本文提出的算法总能找到运行时间更小的解. 在相同的代码大小约束下,我们的算法得到的程序运行时间最多可比贪婪算法减少 18.7%,平均减少 9.4%.

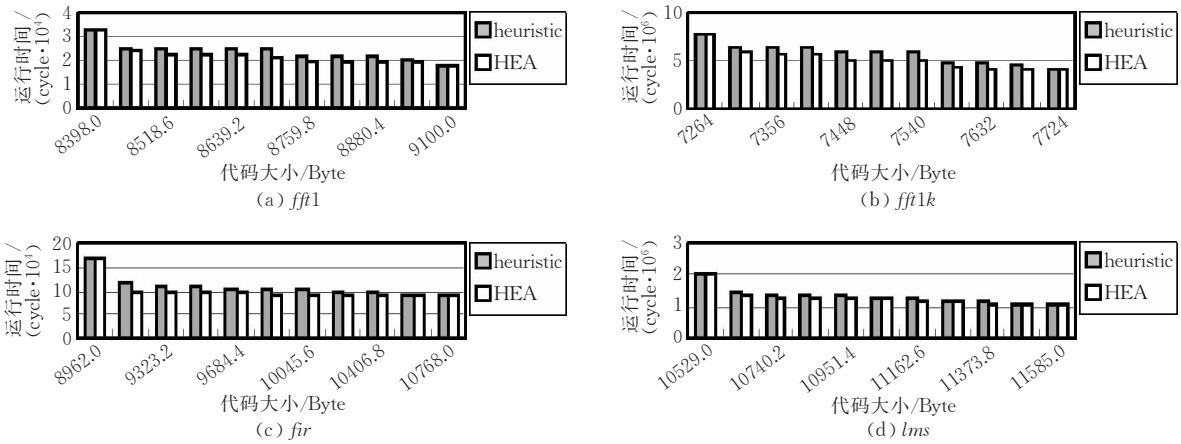

图 1 混合演化算法和贪婪算法在存储空间约束条件下对程序运行时间的比较

虽然本文的讨论以函数为粒度单位,但本文的算法也适用于以程序基本块为单位.

6 结论和将来的工作

嵌入式系统常需要同时满足性能、存储空间、功耗等多个相互冲突的目标,这增加了嵌入式系统编译器代码生成的复杂性. 元启发式算法已被成功地应用于求解很多 NP-hard 组合优化问题,尤其是基于群体的演化算法和蚁群优化算法特别适合求解多目标优化问题,这启发了我们把元启发式算法应用到嵌入式领域的编译优化技术中来.

本文建议了一个可灵活地权衡程序代码大小和性能的演化算法,为双指令集处理器生成混合指令集代码. 此算法基于遗传算法和关键事件禁忌搜索相融合,前者用于全局控制,后者用于局部搜索. 指令选择以函数为单位. 本文还给出了程序总运行时间的估算方法,这依赖于对程序动态行为的 profiling 分析.

我们针对 ARM/Thumb 处理器实现了所建议的算法,并在一组 benchmark 程序上做了实验. 实验结果表明,通过把程序的不同部分选择性地编译为不同指令集的代码,本文的算法能有效地权衡程序的代码大小和运行时间. 相对于贪婪简单启发式算法,在给定相同的代码大小约束条件下,本文的算法总能得到运行时间更少的权衡方案.

本文的工作还需进一步完善. 首先,相对于函数级的粗粒度指令选择,基于基本块甚至指令级的指令选择算法期望能获得更好的结果. 第二,可扩展到其它优化目标(例如功耗). 第三,应用其它多目标优化方法,为嵌入式领域的编译优化提供更好的灵活性. 最后,我们还将研究双指令集处理器的寄存器分配算法,因为合理组织编译器各阶段的耦合常能提高编译器生成代码的质量.

参 考 文 献

[1] Furber S B. ARM system architecture. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 1996

[2] Kirkpatrick S, Gelatt C D, Vecchi M P. Optimization by simulated annealing. Science, 1983, 220(4598): 671-680

[3] Glover F, Laguna M. Tabu Search. Norwell, MA, USA: Kluwer Academic Publishers, 1997

[4] Deb K. Multi-objective optimization using evolutionary algorithms. West Sussex, England: John Wiley&-Sons, 2001

[5] Dorigo M, Stützle T. Ant Colony Optimization. Cambridge, MA, USA: The MIT Press, 2004

[6] Halambi A, Shrivastava A, Biswas P, Dutt N D, Nicolau A. An efficient compiler technique for code size reduction using reduced bit-width ISAs//Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE2002). France, 2002: 402-408

[7] Krishnaswamy A, Gupta R. Profile guided selection of ARM and Thumb instructions//Proceedings of the 2002 Joint Conference on Languages, Compilers, and Tools for Embedded Systems Software and Compilers for Embedded Systems (LCTES2002-SCOPES2002). Berlin, Germany, 2002: 56-64

[8] Lee S, Lee J, Min S L, Hiser J, Davidson J W. Code generation for a dual instruction set processor based on selective code transformation//Proceedings of the 7th International Workshop, Software and Compilers for Embedded Systems, SCOPES 2003. Vienna, Austria, 2003: 33-48

[9] Deb K. A fast and elitist multiobjective genetic algorithm: NSGA-2. IEEE Transactions on Evolutionary Computation, 2002, 6(2): 182-197

[10] Glover F, Kochenberger G. Critical event tabu search for multidimensional knapsack problems//Osman I H, Kelly J P. Meta-Heuristics: Theory and Applications. Norwell, 1996: 407-427

[11] Smith M D, Holloway G. An introduction to Machine-SUIF and its portable libraries for analysis and optimization. The Machine-SUIF Documentation Set. Harvard University, 2002

WU Sheng-Ning, born in 1971, Ph. D. candidate. His research interests include compiler techniques for embedded systems, operating systems, and artificial intelligence.

LI Si-Kun, born in 1941, professor, Ph. D. supervisor. His research interests include electronic CAD, chip design methodologies, and virtual reality.

Background

Meta-heuristics can overcome the limitations of classic optimization algorithms and can search the solution space systematically to solve many optimization problems. There are a variety of optimization problems in the compiler domain. These problems are more complex in the embedded domain than in non-embedded environments, and make standard compiler techniques not easily applicable. Meta-heuristics open up a broad way to solve complex compiler optimization problems.

This research is supported by the National Natural Science Foundation of China, Techniques of Mapping from Information Processing Algorithms to SOC, under grant No. 90207019. This project is to boost the development of the design methodologies of System on Chip (SOC) and electronic design automation.

The research group has been working on the synthesis, validating, and testing of SOC since 2002. This paper proposes a code generation technique for dual instruction processors, which providing compilation technique support for embedded system based on SOC.