Metric 多核子方法划分编译算法设计与实现

傅忠传[1], 高 洋[1], 李 东[1], 张泽旭[1,2], 崔平远[2], 李馨梅[3]

(1. 哈尔滨工业大学 计算机科学与技术学院, 150001 哈尔滨, fuzhongchuan@gmail.com;
2. 哈尔滨工业大学 航天学院, 150001 哈尔滨; 3. 东北农业大学 理学院, 150030 哈尔滨)

摘 要: 为解决编程模型严重阻碍多核处理器性能的进一步提升,尝试将编程单位即方法(procedure)引入到多核处理器结构设计中,提出了面向高性能计算的 Metric(Method Centric)以方法为中心多核架构且采用编译技术,将函数划分成大小相当的子方法(slice),以支持处理器微结构设计. 实验表明: 子方法划分算法的突破,将为 Metric 多核完整编译工具链的建立和模拟器的编写奠定基础.

关键词: 多核;编程模型;编译技术

中图分类号: TP391.9 **文献标志码:** A **文章编号:** 0367-6234(2011)07-0076-04

Design and implementation of slice partition compilation algorithm for Metric multi-core

FU Zhong-chuan[1], GAO Yang[1], LI Dong[1], ZHANG Ze-xu[1,2], CUI Ping-yuan[2], LI Xin-mei[3]

(1. School of Computer Science and Technology, Harbin Institute of Technology, 150001 Harbin, China, fuzhongchuan@gmail.com; 2. School of Astronautics, Harbin Institute of Technology, 150001 Harbin, China; 3. College of Science, Northeast Agricultural University, 150030 Harbin, China)

Abstract: Programming model has become one of the handicaps in multi-core era. To counter to this bottleneck, a novel multi-core architecture, namely Metric (Method centric), is proposed in this paper, which attempts to bring the concept of procedure in programming model into computer architecture design. Basing on Metric multi-core, a slice partitioning compilation pass which tries to divide procedures into finer grains of similar size, namely slice, is investigated in details. Preliminary experiment results validate its effectiveness, and the algorithm is of great importance to Metric compiler tool chain and simulator setup.

Key words: multi-core; programming model; compilation

编程模型严重阻碍了多核处理器性能的进一步提升,为解决编程模型瓶颈,尝试将编程单位即函数或方法(procedure),引入到多核处理器设计中并提出 Metric(Method Centric)以方法为中心多核架构. 由于函数大小不同,不便于处理器微结构设计,为此,采用编译技术将方法划分为大小相近的子方法(slice). 子方法内部指令按数据流方式执行,以最大程度地发掘应用的并行性. 以子方法为粒度执行,将为以方法为粒度的高层优化机制的发掘奠定基础.

本文基于 GCC 编译工具链,对子方法划分方法进行深入研究. 首先对子方法划分原则进行深入探讨;在此基础上,对算法进行详细设计与描述;最后,通过测试验证了本算法的有效性.

1 相关背景

工艺与处理器结构设计的矛盾已引起国际学术界的密切关注,RAW 等[1-2]进行了初步尝试. 近年,在面向高性能计算的瓦式(Tiled Architecture)多核领域国际上竞相展开研究,其中以美国的 WaveScalar 和 TRIPS 模型、欧洲航天计划自适应计算组的微线程模型(Microthread)最具代表性.

收稿日期: 2010-05-26.
基金项目: 国家自然科学基金资助项目(90818016);中国博士后科学基金资助项目(20070420868);黑龙江省自然科学基金资助项目(F200822).
作者简介: 傅忠传(1970—),男,副教授,硕士生导师;
崔平远(1961—),男,教授,博士生导师.

得克萨斯大学奥斯汀分校的 TRIPS 项目(Tera-op, Reliable, Intelligently adaptive Processing System)[3]和华盛顿大学的 WaveScalar[4]是学术界面向高性能计算的瓦式多核的代表，其目标为可升级的、具有万亿次计算能力的通用单片超级计算机(Single Chip Supercomputer).

TRIPS 结合了数据流和超标量的优点，以编译器划分的超块(Hyper block)为单位进行调度和执行. TRIPS 采用 EDGE (Explicit Dataflow Graph Execution)指令集来直接表达数据相关性，即在超块内部为数据流驱动执行[5].

WaveScalar 是一个更加单纯的数据流瓦式多核，以编译器划分的 Wave 为单位调度和执行. WaveScalar 通过简单流水提高性能，没有采用任何前瞻技术[6].

TRIPS 和 WaveScalar 均基于现有的编程模型，在编译器的支持下实现.

阿姆斯特丹大学提出了 MicroGrid 计划[7]——基于微线程模型的瓦式多核结构，并已成为欧洲 AETHER 航天计划中自适应计算重要组成部分[8]. MicroGrid 对现有编程模型进行了丰富和扩展，以 C 语言为基础提出了 μTC，增加了并行编程要素，使程序员在源码级直接(explicitly)表达并行性，并以微线程(Microthread)为手段动态调度和执行. 微线程内部指令仍采用数据流方式执行，即其本质仍然是数据流.

综上，面向高性能计算瓦式多核研究特点为：

1) 无论是 TRIPS、WaveScalar，还是 MicroGrid，均采用软硬协同的设计方式，在编译支持下将应用划分成为粗粒度的指令块——超块、Wave 或者微线程，并以指令块为单位调度和执行.

2) 为最大程度地挖掘应用的并行性，在指令块内部，无论是超块、Wave 或者微线程，均按照数据流驱动方式执行，即从本质上来说都采用数据流执行模型来解决并行度问题.

3) 在编译支持下将应用划分成粗粒度的指令块，使处理器微结构部件的设计不依赖于全局性信息，为核心部件的分布式设计，突破可升级瓶颈提供可能.

近年来，多核在航空、航天和国防武器系统中的应用，引起了国际学术界和厂商的高度关注. 总之，无论在商业、航空、航天，以及国防领域，多核的应用已成为历史必然.

2　Metric 多核架构

纵观国际上各种流行的处理器结构，大都以指令为单位执行，TRIPS 以超块为单位执行，WaveScalar 以其专用的数据流指令为单位执行、以 Wave 为单位调度，MicroGrid 则以微线程为调度单位. 但在编程模型中，函数或方法(Function/Method)却是程序员编写代码的基本单位，这种编程模型与执行模型的不一致性称之为“方法缺陷”.

多核时代 memory wall 和 scaling wall 产生的根本原因在于编程模型与执行模型的不一致性. 为实现编程模型与执行模型的统一，将编程模型中的函数，称之为方法，引入到多核处理器结构、微结构设计中，提出以方法为中心多核架构，试图使执行模型与编程模型相统一.

由于方法——即函数大小不同，不便于处理器微结构设计，采用编译技术将其划分为大小相当的子方法(slice). 该版本的以方法为中心多核处理器称为 Metric，也就是说 Metric 特指是以 slice 子方法为单位执行的多核处理器.

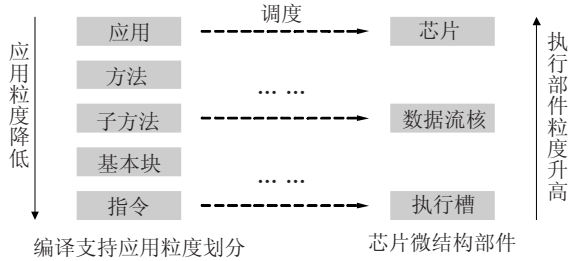

图 1　Metric 多核架构示意图

3　Slice 子方法划分编译技术研究

通过对 GCC 的修改，为 Metric 提供编译支持，为以方法为中心模拟器的建立与验证奠定基础. 编译工具链整体方案如图 2 所示.

设计中应将 GCC 中原有的集中式寄存器文件替换为分布式寄存器文件，且编译中无需 un-SSA. 基于函数的 cfg 图，将函数划分成能大小相当的细粒度的 slice 子方法，slice 由多个基本块组成.

如图 2 中阴影所示，编译链完成为：函数识别与 cfg 调整、访存指令收集、slice 划分、访存指令局部性年龄编码建立、slice 数据流化与优化，以及 slice 管理等功能.

函数识别将函数区分为库函数与用户自定义函数 2 种，并分别进行不同处理. 之后，生成函数调用图(call graph)，并根据函数调用图进一步将用户定义函数分为 2 种情况：被库调用和不被库调用. 被库调用的用户自定义函数，仍然被标记为“系统”，采用系统库的处理方式；不被库调用的用户定义函数则被标记为“用户”，进行函数内联 inline 与优化等处理.

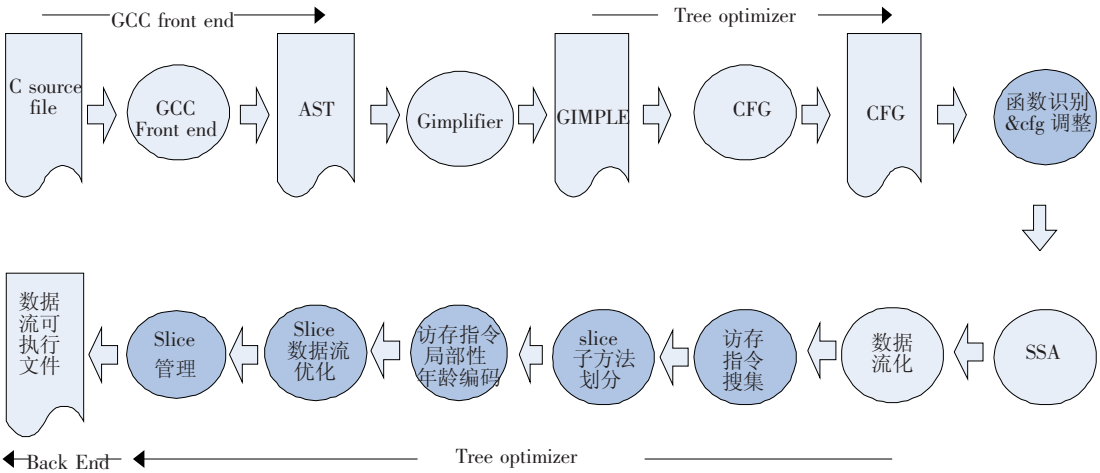

图 2 Metric 多核编译工具链总体方案

编译链以函数为单位对访存指令进行搜集，为访存指令局部性年龄编码建立奠定基础.

Metric 多核处理器以编译划分的 slice 子方法为单位执行和调度，1 个 slice 就如同 1 条粗粒度的指令一样. slice 划分将函数划分成细粒度的子方法、slice 数据流化，保证了 slice 内部指令按照数据流方式执行，以解决并行度问题. 优化用以降低编译代价. 其中，子方法划分已成为本文讨论的重点.

3.1 slice 划分原理

编译链基于函数 cfg 将函数划分成细粒度的 slice 子方法，slice 是一串连续不包含循环且具有单一入口的相关指令的序列.

子方法类似于 WaveScalar 中的 Wave 或 TRIPS 中的超块，从形式上来说它是函数中互联的、单入口、非循环的有向流图的一部分. 子方法中包括多个基本块，基本块包括多条指令. slice 划分原则为：

1）单入口控制流；

2）slice 最多包含 SLICE_MAX 条指令；

3）slice 中的每条指令最多只能执行 1 次；

4）slice 从函数入口基本块开始划分；

5）函数中的循环结构与函数调用，被分成独立的 slice；

6）slice 划分以函数为单位，不能跨越函数，单个 slice 最大为 1 个函数的所有指令；

从本质上来说，slice 子方法划分的目的是抽取数据相关链的最小集合，即使 slice 在关键路径上.

3.2 slice 划分算法

以函数为单位抽取 cfg 流图后，以基本块为单位进行 slice 划分. slice 划分过程为：算法首先先深遍历函数的 CFG 图，遇到循环开始、函数调用、函数出口基本块停止遍历. 之后，遍历结果集中除 slice 头基本块（slice header block）之外的所有基本块，判断其前驱基本块是否全部在先深遍历结果集中，并删除前驱不全在结果集合中的基本块. 循环开始、函数调用、函数出口基本块做为 slice 头节点开始新的划分. 算法具体描述为：

1）将函数入口基本块压入 workList 栈；

2）如果 workList 栈不为空，弹出栈顶基本块，转向步骤 2）；如果 workList 栈为空，本函数 slice 划分完成，结束；

3）如已经对基本块 basicblock 完成划分，转向步骤 2），否则转向步骤 4）；

4）标志 basicblock 为 slice 头节点，将其压入 slice_edge_stack 栈中；

5）如果 slice_edge_stack 不为空，弹出栈顶基本块 Pblock 链，转向步骤 6）；如果 slice_edge_stack 为空，转向步骤 11）；

6）判断该基本块是否为：

条件 1：函数调用基本块

条件 2：函数结束基本块

条件 3：已经被划分为 slice

条件 4：循环开始

上述条件中的任 1 条件成立则转到步骤 5），否则转向步骤 7）；

7）标记 Pblock 是以 basicblock 为头节点的 slice 划分的一个成员，转向步骤 8）；

8）遍历 Pblock 基本块的出口基本块链表，将所有既不是循环开始也不是循环结束的出口基本块压入 slice_edge_stack；

9）将 Pblock 链接到以 basicblock 为头节点的 slice 链表的尾部，跳转至步骤 5）；

10）遍历 slice 链表中的所有基本块，如某基

本块有多个入口基本块，且这几个入口基本块并不是完全都包含在该 slice 链表中，则从 slice 链表中删除基本块 B 及其后的所有基本块；

11）完成 slice 划分，将 slice 链表指针存入头节点基本块 basicblock 中；

12）遍历 slice 链表，如出口基本块多于 1 个，将不包含在该 slice 链表里的所有出口基本块压入 workList 栈，跳转至步骤 2）；

slice 划分使用了 workList 和 slice_edge_stack 2 个栈结构. 函数中的所有基本块形成 1 个双向链表，并对函数入口基本块和出口基本块进行标记.

基本块是构成 slice 的基本单位，同一 slice 中的基本块形成双向链表. 数据结构包括相应标记，标识是否为 slice 首基本块，基本块属于哪个 slice 等信息. 属于某基本块的所有指令构成双向链表.

3.3　实验结果

本文硬件测试平台配置为酷睿 2.0 GHz、内存 2 GB，硬盘 20 GB，软件环境为 Red Hat Linux 7.3.

采用 6 个测试程序，包括 Mibench 测试集中的 dijkstra 和 susan，自编写 4 个测试基准为：maxscore、score-sort、lcs 和 math. Maxscore 和 score-sort 针对数组操作完成最大值超找和排序，LCS 完成字符串操作，math 为数学运算的代表，dijkstra 和 susan 等针对网络应用，采用小输入集.

sice 划分算法初步实验结果如表 1 所示，显示了 slice 中平均指令数和 slice 内的基本块数. 可见，当程序中函数数较少时，slice 划分的基本块个数较少（≤3），这造成 slice 中指令数较少，因此无法充分发挥处理器的性能. 程序中函数个数较多时，slice 划分效果更好.

表 1　slice 中基本块分布

测试基准名称	slice 数量	函数数量	slice 中基本块数					
			1	%	2	%	≥3	%
susan	69	7	11	16	24	35	34	49
dijkstra	237	17	39	16	79	34	121	50
lcs	184	16	31	17	67	36	86	47
score-sort	33	4	13	39	17	52	3	9
math99	16	2	6	38	10	62	0	0
maxscore	4	1	2	50	2	50	0	0

4　结　论

1）为解决编程模型瓶颈，尝试将编程模型中的函数引入到多核处理器设计中，提出 Metric（Method Centric）以方法为中心多核架构.

2）由于函数大小不同，不便于处理器微结构设计，为此采用编译将方法划分大小相近的子方法（slice）. 给出 Metric 编译工具链设计方案.

3）对子方法划分方法进行深入探讨，并给出算法描述.

4）采用典型测试基准对算法划分效果进行初步实验，实验结果表明算法的有效性.

参考文献：

[1] WAINGOLD E, TAYLOR M, SARKAR V, *et al.* Baring it all to software: RAW machines[J]. IEEE Computer, 1997, 30(9):86 - 93.

[2] MAI K, PAASKE T, JAYASENA N, *et al.* Smart memories: A modular reconfigurable architecture[C]//Proceedings of the 27th Annual International Symposium on Computer Architecture. New York, NY: ACM, 2000: 161 - 171.

[3] University of TEXAS at Austin. Fera-op Reliable Intelligently adaptive processing system [EB/OL]. [2010 - 04 - 19]. www. cs. utexas. edu/ ~ TRIPS/.

[4] University of Washingon. WaveScalar [EB/OL]. [2010-04-19]. http://wavescalar. cs. washington. edu.

[5] SANKARALINGAM K, NAGARAJAN R, McDONAID R, *et al.* Distributed microarchitectural protocols in the trips prototype processor[C]//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, DC: IEEE, 2006: 480 - 491.

[6] SWANSON S, PUTNAM A, MERCALDI M, *et al.* Area-performance trade-offs in tiled dataflow architectures [C]//Proceedings of the 33rd Annual International Symposium on Computer Architecture. Washington, DC: IEEE, 2006: 314 - 326.

[7] University of Amsterdam. MultiProcessor System-on-Chip (MP-SoC) Design[EB/OL]. [2010 - 04 - 19]. http://www. science. uva. nl/research/csa/.

[8] BELL I, HASAASNEH N, JESSHOPE C. Supporting microthread scheduling and synchronisation in CMPs [J]. Intl J Parallel Processing, 2006, 34(4): 1 - 9.

（编辑　张　红）