# 程序阶段性分析和阶段检测技术

张海博 安 虹 贺松涛 孙 涛 王 涛 彭 毅 程亦超
(中国科学技术大学计算机科学与技术学院 合肥 230027)

**摘 要** 对称多处理器的飞速发展和近年来提出的动态异构处理器(DHMP)为性能优化提供了新的机遇。一个机遇是找出程序每个阶段的性能瓶颈,提出了静态程序阶段分析方法,即通过分析结构参数和计算相似度矩阵来找出程序每个阶段的资源瓶颈;另一个机遇是给出动态异构处理器重构的时间节点,提出了DPDA和HTPD两种动态阶段检测算法,检测出阶段的变化能够为动态可重构处理器提供重构的时间节点。DPDA算法效果很好且软硬件实现代价小,而HTPD算法是目前为止第一个使用统计学方法进行动态检测阶段的算法。实验表明,与BBV相比,DPDA和HTPD能避免BBV离线、动态算法需添加额外硬件、结果与编译器相关等限制,并且阶段划分的稳定性和正确率与BBV相当。DPDA和HTPD算法由于本身不依赖额外硬件,因此都能直接在主流处理器和动态异构处理器(DHMP)中使用。

**关键词** 程序分析,程序阶段性,静态程序分析,阶段检测
**中图法分类号** TP368.1 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.1.016

## Program Phase Analysis and Phase Detection Techniques

ZHANG Hai-bo AN Hong HE Song-tao SUN Tao WANG Tao PENG Yi CHENG Yi-chao
(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

**Abstract** The rapid development of SMP and new proposed DHMP bring new challenges for program performance optimization. We raised two performance tuning problems and the solutions were given by phase analysis. The first problem is to find theperformance bottlenecks in each phase. We proposed a static phase analysis method, which finds performance bottlenecks in each phases by analyzing architecture features and its similar matrix. The second problem is to give the proper time to reconfigure for DHMP. We proposed dynamic phase detection algorithms, namely DPDA and HTPA. DPDA archives effective performance in a relative low software/hardware cost, and HTPD is the first phase detection algorithm using statistics theory. Our results show that comparing with BBV, DPDA and HTPD can avoid its limitation of offline, plus additional hardware in online algorithm and compiler's effect, while they offer a comparable stability and correctness. Since DPDA and HTPD do not relay on additional hardware support, they can be implemented directly in mainstream processors and DHMP.

**Keywords** Program analysis, Program phase, Static program analysis, Phase detection

## 1 引言

对称多处理器的飞速发展和近年来提出的动态异构处理器为单线程程序分析带来两个机遇。第一个机遇是:传统微处理器设计目标是关注如何优化程序的平均性能,导致程序在每个阶段的性能瓶颈不一定相同。程序分析可以指出每个阶段的性能瓶颈。另外,动态异构处理器(DHMP)的提出给串行程序分析和优化带来新的机遇。IBM Power7, Intel SandyBridge等新型处理器能够在片上集成多个同构处理器核心[1,2]。但这种处理器面临这样的矛盾:若芯片上全放置少数复杂的乱序超标量核心,就无法保证高吞吐量的并行负载;另一方面,如果芯片上全部放置大量顺序单发射核心,关键区域代码的性能就会急剧下降。为了解决这样的问题,CoreFusion, WiDGET, TFlex等便顺势提出DHMP[3-5]。DHMP的好处是可根据程序对资源的需求对硬件动态可重构,从而在能耗和性能上都趋于最优。因此另一个机遇是:面对这样的动态可重构结构,需要找到合适的时间节点进行结构重构。

本文提出使用程序阶段性分析技术来解决上述的问题。首先分析影响性能的资源瓶颈,将其抽象为一组结构参数。

到稿日期:2013-12-26 返修日期:2014-03-15 本文受国家自然科学基金(60970023),国家973计划项目(2011CB302501),国家863计划项目(2012AA010902,2012AA010901)资助。

张海博(1989-),男,硕士生,主要研究领域为多核众核芯片体系结构、程序分析和性能建模;安 虹(1963-),女,博士,教授,主要研究领域为计算机系统结构和并行处理器体系结构,E-mail:han@ustc.edu.cn(通信作者);贺松涛(1992-),主要研究领域为程序分析;孙 涛(1987-),男,博士,主要研究领域为众核结构系统的性能优化;王 涛(1988-),男,硕士,主要研究领域为多核众核芯片体系结构、异构平台任务调度;彭 毅(1990-),男,硕士生,主要研究领域为片上网络;程亦超(1989-),男,硕士生,主要研究领域为并行算法分析和优化。

用片上性能检测单元(PMU)采集 SPEC2000 程序各结构参数,比较各采样点的相似度,可以静态地划分程序阶段,并进一步得出结构参数和性能之间的关系。提出了两种阶段检测算法 DPDA 和 HTPD,这两种阶段检测算法能够动态检测程序阶段变化。实验表明,这两种阶段检测算法的正确率和稳定性与 BBV 技术相当,而与代码、输入和编译器无关,有更好的适用性,能够在动态异构平台上使用。

## 2 相关工作和动机

T. Sherwood 等人利用基本块分布分析来探测程序的阶段性[6-8]。程序的基本块分布向量(Basic Block Vector,BBV)记录了一段特定的程序执行区间中的各基本块执行频率。当两个相邻 BBV 的曼哈顿距离达到某个预先设定的阈值时,就可以唯一地确定程序中不同的程序阶段。Dhodapkar 等人对条件分支分析技术、工作集距离分析技术和 BBV 分析技术进行比较[9]。他们认为,BBV 虽然提供了更好的敏感度,与性能的变化也能保持近似,但不足之处在于 BBV 技术的稳定性稍差,这样会带来更多的错误率,也会引起频繁的结构重构。

在我们看来,包括 BBV 在内的代码级分析技术还存在一些明显缺点。首先,动态异构处理器在运行时无法知道代码级阶段变化,无法对结构重构作出准确的判断。另外,BBV 能够通过分析得到程序最有代表性的部分,然而动态重构平台想要得到的却是由于结构上的何种因素影响了程序的性能。另外,BBV 技术不能在线地得出阶段的划分,也会由于程序输入的不同而改变阶段点。我们认为代码级程序分析技术在动态异构平台上已经不再适用,需要提出新的结构资源感知的程序阶段性分析技术以能够检测出程序性能的变化。

## 3 程序性能瓶颈分析

首先解决第一个问题,即指出每个程序阶段是由何种片上资源瓶颈限制了程序的性能。对于大多数通用处理器,影响指令顺利提交的主要因素分为几类:错误的分支指令推测、指令缓存缺失、乱序发射部件资源限制、访存导致的处理器停顿。这 4 类影响程序性能的指标都可以通过增加相应部件得到缓解。例如,增加末级缓存大小可以减小缓存缺失率,从而减少访存导致的处理器停顿,进而增加性能。选取表 1 所列的结构参数作为分析的重点。

表 1 选取的结构参数和性能检测事件公式

| 结构参数 | 性能检测事件公式[1] |
|---|---|
| 核内缓存停顿 | STALLS_LDM_PENDING−CYCLES_L2_PENDING |
| 核外缓存/访存停顿 | CYCLES_L2_PENDING |
| 乱序执行部件停顿 | RESOURCE_STALLS. ANY |
| 推测执行错误停顿 | (UOPS_ISSUED. ANY−UOPS_RETIRED. RETIRE_SLOTS+4 * INT_MISC. RECOVERY_CYCLES )/4 |

## 4 程序阶段性静态分析

我们的实验平台是 Intel SandyBridge 架构的 i5-2400 处理器[1],使用 10Mcycles 频率采集结构参数。选取了具有代表意义的 10 个 SPEC2000 程序/输入对作为本文的基本测试程序(见表 2)。图 1 显示了考虑结构参数的处理器停顿分布。可以发现,每个程序的结构参数对处理器的影响是不同的。需要指出的是 ccl/1 和 gzip/p,我们发现这两个程序的 IPC 很接近,但它们对资源的需求是不同的。假设能将 gzip/p 程序放置在核内缓存较大的处理器上,则其相比 ccl/1 程序的加速会更明显。

表 2 实验所选基准测试程序

| SPEC 程序 | 程序输入 |
|---|---|
| ammp | ammp. in |
| bzip | input. graphic |
| bzip | input. program |
| cc | 166. i |
| cc | scilab. i |
| gzip | input. graphic |
| gzip | input. program |
| mcf | inp. in |
| perl | diffmail. pl |
| perl | splitmail. pl |

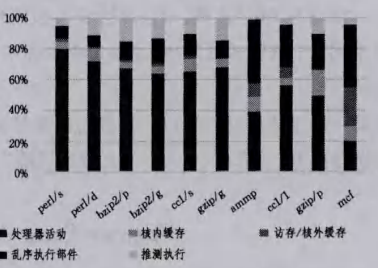


图 1 各程序处理器停顿分布

为了静态划分程序阶段,使用结构参数相似性矩阵(AFSM)来表达两个采样区间的相关程度,并使用曼哈顿距离来量化两个区间的距离。将得到的相似性矩阵归一化,得到一个区间为[0,1]的三角阵。三角阵元素值越接近于 0,则认为两个区间越相似;反之认为这两个区间越不相似。图 2 所示的就是 gzip/g 的相似度矩阵。为了凸显阶段的不同,再把该矩阵变换到灰度矩阵上,获得其灰度图。这里 PMU 对整个 gzip/g 执行过程进行了 4926 次采样。比较不相邻两个区间的相似度,该矩阵将 gzip/g 分为 10 个阶段。

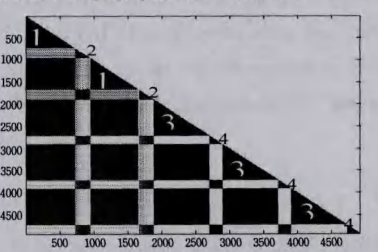


图 2 gzip/g 的相似度矩阵

## 5 程序阶段性动态分析

### 5.1 DPDA(Direct Phase Detection Advanced)

程序阶段检测算法试图寻找那些可以被利用的主要阶段变化。我们意识到动态重构需要时间开销,频繁的阶段划分导致重构会降低系统效率。所以动态阶段侧重于主要阶段变化,减少不必要的阶段划分。为了避免扰动产生的阶段被错误划分,用计数器记录连续判断出的阶段变化等于 $k$ 时,认为发生阶段变化。在这个算法中,$k$ 的选择不宜过大,否则造成阶段划分延时,甚至造成阶段无法识别。我们称这个算法为 DPDA。

DPDA 算法伪代码

A 表示各个程序区间数组,Last 表示上一个程序阶段划分的位置,K

表示当前位置，Threshold 为阈值。

```
A_old = mean(A(Last:K-1));
A_now = A(K);
Delta=abs(A_now / A_old - 1);
if(Delta > threshold )
    Counter++;
else
    Counter = 0;
if(Counter == k)
    Threshold = UPPERBOUND;
    Last = K;
    Counter = 0;
else
    Threshold = Threshold - AGINGRATE;
    Threshold = MAX(LOWEROUND,Threshold);
K=K+1;
```

### 5.2 HTPD(Hypothesis Testing Phase Detection)

我们还提出了一种新的算法，将其命名为 HTPD。这个算法的思想主要借鉴了在数理统计学中的假设检验问题。如果两组样本来自均值相同的两个正态分布总体，那么将样本数据代入式(1)后得到的结果应该服从标准正态分布。反之，如果得到的结果与 0 的偏离量很大，那么可以拒绝之前对两个正态分布总体均值相等的假设，即得到两个正态分布总体均值不等的判定。

$$\frac{(\overline{X}-\overline{Y})}{\sqrt{S_x^2/n+S_y^2/m}} \tag{1}$$

HTPD 算法伪代码：

A 表示各个程序区间的参数数组，Last 表示上一个程序阶段划分的位置，K 表示当前位置，threshold 为阈值，n，m 与算法描述中一致，下界选为 5%。

```
temp_n = Min(n,k-m-last-1);
A_1 = A(K-m-temp_n:K-m-1));//第一组样本
A_2 = A(K-m+1:K);//第二组样本
M1 = mean(A_1);M2 = mean(A_2);
S1 = std(A_1);S2 = std(A_2);
T=(M1 - M2)/ (S1^2 / temp_n + S2^2 / m ) ;
if( abs(T) >threshold &&(M1-M2)/ M2 > 0.05 )
    Last = K;//进行阶段划分
    K = K+m;
else
    K = K+1;
K=K+1;
```

提出的 HTPD 算法就是借鉴了上述假设检验的思想。选取最近得到的 $n+m+1$ 个数据，将其分成两组，前 $n$ 个为一组，后 $m$ 个为另一组。现在判断是否在第 $n+1$ 个程序区间上发生了变化，即判断第一组和第二组是否处于同一个阶段。把这两组数据代入式(1)，当其结果大于一个给定的阈值时，即判定这两组数据属于不同的阶段。另一方面，式(1)中的标准差部分很小时，均值差的轻微变化都可能大于阈值。然而，微小扰动造成的阶段划分在实际中是没有意义的，完全可以将其视为同一个阶段，因此需要一个划分阶段的下界，即阶段划分至少需要满足两组样本的平均值之间的相对差异大于一个下界值。这个机制可以使阶段划分更加稳定，更加侧重于主要阶段变化。

### 5.3 实验结果

实验的评测指标有两个，一个是由 Jeremy Lau 等人提出，将各个阶段内的变异系数(CoV)加权平均的结果作为评测指标[10]；同时，阶段划分的稳定性也是一个很重要的因素，因此使用程序阶段平均长度作为另一个评测标准。需要指出的是 DPDA、HTPD 均可以动态地实现。

图 3 指出，HTPD 算法在平均阶段长度中表现较好。这是由于相对 DPDA 算法，HTPD 算法不仅仅考虑了平均值的变化，还考虑了方差造成的影响，当数据波动较大时结果也会增大，从而避免波动干扰或者小的阶段产生的影响；另一方面，由于 HTPD 对微小的阶段并不敏感，其划分出的较大阶段包含了很多微小的阶段，导致各个阶段的变异系数较大，造成了较大的 CPI-CoV。因此，HTPD 算法更侧重于长度较长的主要阶段变化，对微小的阶段或干扰具有较好的过滤作用。还可以发现，DPDA 算法在 CPI-CoV 上比 HTPD 有一定优势。

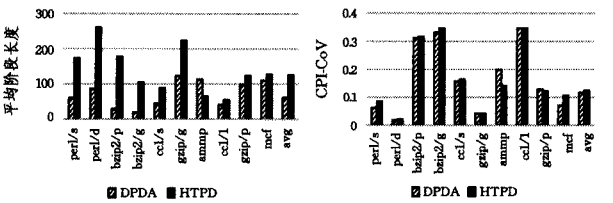


图 3 各阶段检测算法平均阶段长度和 CPI-CoV 比较

进一步地，将 HTPD 算法、DPDA 算法和 BBV 算法进行比较。在 Jeremy Lau 等人的工作中[10]，对 BBV 算法在 SPEC2000 工作集上的 CPI-CoV 进行了实验测量，我们的输入集也是相同的，可以直接使用其结果进行比较。通过图 4 可以发现，3 个算法的 CPI-CoV 指标相当，可以说明 HTPD 算法和 DPDA 算法在准确性和稳定性方面与 BBV 算法是相当的。同时由于 BBV 算法是静态的阶段检测算法，其需要采集的值无法通过 PMU 采集，需要添加额外硬件。从这个角度看，HTPD 算法和 DPDA 算法更具有优势。

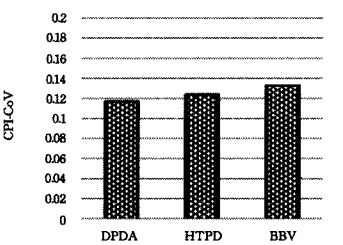


图 4 阶段检测算法 CIP-CoV 比较

**结束语** 本文利用程序阶段分析技术对两个问题给出了解决方案，即能通过分析结构参数获知一个阶段程序的性能瓶颈，且能检测出何时发生阶段变化。所提出的静态方法通过分析结构参数和计算相似度矩阵找出程序每个阶段中的资源瓶颈。它可以为编译器代码优化、资源分配算法给出很好的提示。本文提出了 DPDA 和 HTPD 两种动态阶段检测算法，DPDA 算法效果好且软硬件实现代价小，而 HTPD 算法是目前为止第一个使用统计学方法进行动态检测阶段的算法。实验表明，与目前最好的阶段性分析技术 BBV 相比，DPDA 和 HTPD 能避免 BBV 离线、动态算法需添加硬件、结果与输入相关等限制，与 BBV 技术是相当的。

程序阶段性预测是值得研究的工作。利用程序阶段性预

测可以预知下一个阶段的资源瓶颈，能更有针对性地提出资源重构的建议。程序阶段性分析在调度算法中的资源分配应用也是一个值得研究的问题，试想在有多个不同处理器核的动态异构处理器上调度进程，如何根据阶段性技术和资源瓶颈分析进行资源分配优化也需要更多的实验工作。

## 参 考 文 献


[1] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manualys[OL]. http://www.intel.com/content/www.us/en/processors/architectures-software-developer-manuals.html

[2] Kalla R, Sinharoy B, Starke W J, et al. Power7: IBM's Next-Generation Server Processor[J]. IEEE Micro, 2010, 30(2): 7-15

[3] Ipek E, Kirman M, Kirman N, et al. Core fusion: accommodating software diversity in chip multiprocessors[C]// Proceedings of the 34th annual international symposium onComputer architecture. Dean Tullsen ed. San Diego, California, USA: ACM, 2007: 186-197

[4] Kim C, Sethumadhavan S, Govindan M S, et al. Composable Lightweight Processors[C]// Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2007: 381-394

[5] Watanabe Y, Davis J D, Wood D A. WiDGET: Wisconsin decoupled grid execution tiles[C]// Proceedings of the 37th Annual International Symposium on Computer Architecture. André SeznecedSaint-Malo, France, ACM, 2010: 2-13

[6] Sherwood T, Sair S, Calder B. Phase tracking and prediction[C]// Proceedings of the 30th Annual International Symposium on Computer Architecture. New York, NY, USA, ACM, 2003: 336-349

[7] Sherwood T, Perelman E, Calder B. Basic block distribution analysis to find periodic behavior and simulation[C]// Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques. Barcelona, Catalunya, Spain, IEEE Computer Society, 2001: 3-14

[8] Sherwood T, Perelman E, Hamerly G, et al. Automatically characterizing large scale program behavior[C]// Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, NY, USA, ACM, 2002: 45-57

[9] Dhodapkar A S, Smith J E. Comparing Program Phase Detection Techniques[C]// Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, Washington DC, USA, 2003: 217-228

[10] Lau J, Schoenmackers S, Calder B. Transition Phase Classification and Prediction[C]// Proceedings of the 11th International Symposium on High-Performance Computer Architecture. IEEE Computer Society, Washington DC, USA, 2005: 278-289


(上接第 62 页)

速效率角度分析加速性能，实现的并行 AES 算法在 16 核的 Nvidia Geforce G210 上能最高达到 15.83 倍的加速比和 99.898%的加速效率。使用当前普通的显卡(如 Nvidia Geforce G210)加密 SSL 会话数据(大小 35kB～150kB)能提供非常高(>94%)的加速效率。

## 参 考 文 献


[1] Wang Y, Ha Y. FPGA-Based 40.9-Gbits/s Masked AES With Area Optimization for Storage Area Network[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2013, 60(1): 36-40

[2] Mathai A, Sathyanarayana M. Design of Area Optimized AES 128 Algorithm Using Mixcolumn Transformation[J]. International Journal of Innovative Research and Development, 2013, 2(7): 173-176

[3] Prathyusha C, Rani P S. Implementation of Fast Pipelined AES Algorithm on Xilinx FPGA[J]. International Journal of Science and Research, 2013, 2(8): 377-381

[4] Granado-Criado J M, Vega-Rodríguez M A, Sánchez-Pérez J M, et al. A new methodology to implement the AES algorithm using partial and dynamic reconfiguration[J]. INTEGRATION, the VLSI journal, 2010, 43(1): 72-80

[5] Nhat-Phuong T, Myungho L E E, Sugwon H, et al. High Throughput Parallelization of AES-CTR Algorithm[J]. IEICE TRANSACTIONS on Information and Systems, 2013, 96(8): 1685-1695

[6] Cook D, Keromytis A D. Cryptographics: Exploiting Graphics Cards for Security (Advances in Information Security)[M]. Springer, 2006: 78-96

[7] Manavski S A. CUDA compatible GPU as an efficient hardware accelerator for AES cryptography[C]// IEEE International Conference on Signal Processing and Communications. IEEE, 2007: 65-68

[8] Bos J W, Osvik D A, Stefan D. Fast Implementations of AES on Various Platforms[C]// IACR Cryptology ePrint Archive. 2009, 2009: 501

[9] Tomoiagaa R D, Stratulat M. Accelerating Solution Proposal of AES Using a Graphic Processor[J]. Advances in Electrical and Computer Engineering, 2011, 11(4): 99-104

[10] Duta C L, Michiu G, Stoica S, et al. Accelerating Encryption Algorithms Using Parallelism[C]// IEEE International Conference on Control Systems and Computer Science (CSCS). 2013: 549-554

[11] 夏辉，贾智平，张峰，等. AES 专用指令处理器的研究与实现[J]. 计算机研究与发展，2011，48(8)：1554-1562

[12] 向涛，余晨韵，屈晋宇. 基于改进 AES 加密算法的 DICOM 医学图像安全性研究[J]. 电子学报，2012，40(2)：406-411

[13] National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard[OL]. http://www.itl.nist.gov/fipspubs/, Nov. 2001

[14] NVIDIA Corporation. CUDA Technology[OL]. http://www.nvidia.com/CUDA, Sep. 2008

[15] Levering R, Cutler M. The portrait of a common HTML web page[C]// Proceedings of the 2006 ACM symposium on Document engineering. 2006: 198-204