·软件技术与数据库·　　文章编号：1000-3428(2016)01-0051-05　　文献标识码：A　　中图分类号：TP391

基于流访问特征的多级硬件预取

贾　迅，翁志强，胡向东

（上海高性能集成电路设计中心，上海 201204）

摘　要：硬件数据预取技术将处理器可能访问的数据提前装入 Cache 中，使得处理器访存时尽量命中 Cache，提升系统性能。但现有研究和应用主要对一级 Cache 进行预取，预取的数据可能在使用前无法及时装入 Cache，从而降低硬件预取对系统性能的提升效果。针对上述问题，以流访问特征的预取为基础，提出一种同时对多级 Cache 进行预取的方法，并对流访问特征的预取进行实现。基于 SPEC CPU2000 测试程序集的实验结果表明，与仅对一级 Cache 进行预取相比，对多级 Cache 同时进行预取可以将整数程序的性能平均提升 2.11%，最高提升 11.19%，浮点程序的性能平均提升 3.08%，最高提升 12.77%。

关键词：存储墙；流访问；处理器；多级 Cache；硬件预取

中文引用格式：贾　迅，翁志强，胡向东．基于流访问特征的多级硬件预取[J]．计算机工程，2016，42(1)：51-55.

英文引用格式：Jia Xun，Weng Zhiqiang，Hu Xiangdong. Multiple Level Hardware Prefetching Based on Stream Access Features[J]. Computer Engineering，2016，42(1)：51-55.

Multiple Level Hardware Prefetching Based on Stream Access Features

JIA Xun，WENG Zhiqiang，HU Xiangdong

(Shanghai High Performance IC Design Center，Shanghai 201204，China)

【Abstract】 Technique of hardware data prefetching loads data to Cache before they are actually referenced by processor，thus improves the system performance. Existing research and application focus on prefetching data only to the first level Cache. In this way，the prefetched data may arrive at Cache after they are used and decrease the performance of hardware data prefetching. This paper proposes a technique of prefetching multiple levels of Caches based on a stream-based prefetcher. Performance analysis for SPEC CPU2000 shows that the proposed technique can improve the performance of integer applications by 2.11% on average，11.19% at most，float applications 3.08% on average，12.77% at most，compared with only prefetching the first level Cache.

【Key words】 memory wall；stream access；processor；multiple level Cache；hardware prefetching

DOI：10.3969/j.issn.1000-3428.2016.01.010

1　概述

近年来，存储墙问题成为处理器性能提升的主要瓶颈，并且在多核架构下，该问题变得更加严重[1]。为了缓解这个问题，很多技术被提出并使用，其中 Cache 技术应用最为广泛。基于 Cache 的层次化存储结构利用程序的局部性原理，将最近访问的数据装入位于结构上层且访问速度较快的 Cache。在 Cache 命中率较高的情况下，处理器的访存延迟可以近似为访问 Cache 的延迟。而一旦 Cache 访问缺失，处理器必须从低层次的存储结构中访问数据，其带来的访问延迟往往呈数十倍增加，影响了处理器计算性能的发挥。数据预取技术将处理器在未来一段时间内可能访问的数据提前装入 Cache。当处理器访问这些数据时可以直接命中 Cache，缓解了 Cache 访问缺失造成的长延迟问题，提升了处理器的访存性能。

按照实现方式，数据预取技术可分为软件预取和硬件预取。软件预取通过编译器或程序员在程序目标码中插入预取指令实现预取功能。硬件预取通过在处理器的微结构中增加预取引擎，预取引擎根据处理器访存的特征实时生成硬件预取请求。硬件预取对程序员透明，且不存在兼容性问题，因而在商用处理器中得到了广泛应用。目前，绝大部分预取机制的研究和应用仅考虑了对一级 Cache 进行预取。但一级 Cache 的容量有限，使得预取时机的控

基金项目："核高基"重大专项"超级计算机处理器研发"(2013ZX01028-001-001-001)。

作者简介：贾　迅(1989－)，男，硕士研究生，主研方向为计算机系统结构；翁志强，工程师；胡向东，高级工程师。

收稿日期：2015-01-06　　**修回日期**：2015-02-03　　**E-mail**：jia1546@163.com

制成为一个难点:如果预取太早,预取会淘汰 Cache 中即将被访问的数据,造成 Cache 的污染;预取太晚,处理器访存时预取的数据尚未装入 Cache,预取无法有效隐藏访存延迟。

本文针对现代处理器中层次化的 Cache 结构,提出同时对多级 Cache 进行预取的方法,并以流访问特征的预取机制为基础进行实现。

2 相关工作

2.1 基于流访问特征的预取

数据预取技术本质上是对访存序列的预测,因而程序的访存特征是预取引擎工作的前提。其中"流"是一种典型的访问特征,广泛存在于科学计算和工程应用的程序中[2]。流访问特征是指:在一段时间内,程序访存的 Cache 行地址呈现递增或递减的变化规律。假设一段时间内访存的 Cache 行地址为 $C_1\ C_2\cdots C_i\ C_{i+1}\cdots$,定义步长 S 为相邻 Cache 行地址的变化量,即 $S=C_{i+1}-C_i$。通常定义 $S>0$ 的流为向前访问的流,$S<0$ 的流为向后访问的流;并且将 $S=1$ 的流称为"单位步长的流"(unit-stride stream);将 $S>1$ 的流称为"非单位步长的流"(non-unit stride stream)。

针对流访问特征的预取,通常以访问连续的 Cache 行均缺失且缺失次数达到设定的阈值作为识别"流"的依据;如果流访问特征被识别,后续访问流中新的 Cache 行会触发预取。一般通过预取的距离,即预取超前当前访问的 Cache 行数,来调整预取的时机。OBL(One Block Lookahead)方法[3]最早提出,其主要思想是访问一级 Cache 缺失时,预取当前访问 Cache 行的下一个 Cache 行。OBL 方法虽然实现简单,但却容易污染 Cache 且对访存带宽的要求较高。文献[4]为了避免预取数据对 Cache 的污染,在硬件上设置了单独的流缓冲保存预取的数据,并基于流缓冲设计了针对单位步长访问流的预取。文献[5]基于文献[4]提出的流缓冲,研究了对非单位步长的访问流进行预取,并评估了在芯片面积有限的情况下,以流缓冲替代二级 Cache 的可能性。文献[6-7]研究了对步长可变的访问流进行预取的方法,但逻辑及硬件实现较为复杂。此外,主流的处理器厂商如 Intel[8],IBM[9-11]均在其产品中应用了基于流访问特征的硬件预取机制。上述对流访问特征预取的研究和实现仅考虑了对一级 Cache 进行预取,并且为了防止预取对 Cache 的污染,预取距离的设置也较为保守。

2.2 层次化的 Cache 结构

随着处理器计算性能的迅速提升,访问动态随机访问存储器(Dynamic Random Access Memory,DRAM)存储的延迟成为计算机系统的性能瓶颈。采用静态随机访问存储器(Static Random Access Memory,SRAM)设计的 Cache 访问速度较快,但单位比特的成本和面积较大。因此,在处理器和 DRAM 存储之间增加 Cache,并且将最近访问的数据装入 Cache,可以在平均访存延迟、面积以及成本之间取得很好的平衡。为了应对 Cache 访问缺失造成的长延迟问题,现代处理器通常采用层次化的 Cache 结构。层次结构由上至下分别为一级、二级、三级 Cache 等,其中,一级 Cache 的容量和访存延迟较小,二级 Cache、三级 Cache 的容量和访存延迟则依次增加;并且为了便于存储一致性协议的实现,上层 Cache 中的数据一般为下层 Cache 的子集。部分处理器按照 Cache 中存储的内容将一级 Cache 划分为指令 Cache 和数据 Cache,本文后续提及一级 Cache 时均指数据 Cache。以一款国产申威处理器为例,各级 Cache 和主存的容量、访问延迟如表 1 所示。

表 1 申威处理器的存储层次结构

存储层次	容量	访问延迟/周期
一级 Cache	32 KB	4
二级 Cache	256 KB	12
三级 Cache	6 MB	45
主存	–	120

3 多级 Cache 硬件预取

本文基于 SPEC CPU2000 测试程序集[12-13],统计了流预取中访问新的 Cache 行所需周期数,结果如图 1 所示。可以看出,Cache 行之间的访问间隔小于 120 周期的情况占了较大比例。若仅对一级 Cache 进行预取,且访问二级、三级 Cache 均缺失,此时需要从主存中装入数据。由于访问主存的延迟至少为 120 周期,必然导致预取的数据无法在使用前装入一级 Cache,预取隐藏访存延迟的效果有限。

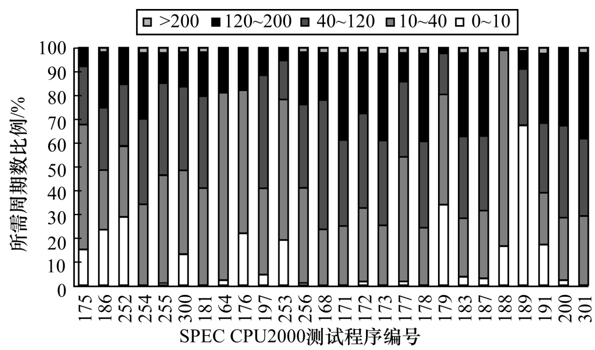

图 1 流预取访问新的 Cache 行所需周期数

本文利用现代处理器中一级、二级、三级 Cache 的容量和访问延迟逐渐增加的特点,同时对三层 Cache 进行预取,并将预取距离较大的请求返回的数据装入较低层次的 Cache 中。在尽量增加预取距离的同时,降低对 Cache 的污染,解决预取时机与

Cache 容量之间的矛盾。

3.1　整体框架

多层硬件数据预取实现的整体框架如图 2 所示，预取引擎记录流水线上访存请求的信息，包括请求的类型、请求访问的 Cache 行地址以及访问是否命中一级 Cache 等。如果预取引擎检测到单位步长访问流，会生成同时对三层 Cache 进行预取的请求。硬件预取请求的格式与普通的访存请求相同，包括请求有效标志位、请求的类型以及请求访问的 Cache 行地址。预取引擎将生成的请求写入预取缓冲，缓冲为 FIFO 结构，并配置 3 个写口，分别用于写入对一级、二级、三级 Cache 进行预取的请求。访存流水线空闲时允许预取缓冲中的硬件预取请求仲裁上流水线执行。如果预取请求命中一级 Cache，说明预取的数据已在 Cache 中，且由 Cache 间的包含关系，预取数据同时也在二级、三级 Cache 中，此时，预取请求被丢弃；否则将请求发往下一级的 Cache 进行后续处理。访问三级 Cache 缺失的请求会访问主存，并将响应数据装填到各级 Cache 中。

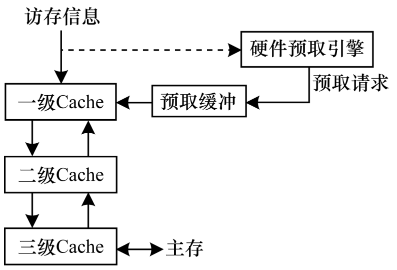

图 2　多层硬件预取实现的整体框架

为了降低逻辑设计的复杂度和硬件开销，本文只针对单位步长的访问流进行预取，并且将预取数据直接装入各级 Cache，而不是装入单独的缓冲。虽然可能造成对 Cache 的污染，但却避免了对预取的数据进行 Cache 一致性处理，简化了设计与实现。

3.2　预取引擎

硬件预取引擎的功能包括流访问特征的识别和预取请求生成。为了识别流访问，硬件上设置流识别缓冲对访存流水线上的请求进行筛选。流识别缓冲记录的信息包括条目的状态、访存的 Cache 行地址、访问缺失的次数，具体的信号和含义如表 2 所示。

表 2　流识别缓冲中条目记录的信息

信号	含义
r_State	INV：无效态 INIT：初始态 TRKFWD：检测向前流访问 TRKBWD：检测向后流访问 PFHFWD：向前访问预取 PFHBWD：向后访问预取
r_Addr	Cache 行地址
r_Time	访问缺失的次数

当访存请求有效时，将请求的 Cache 行地址与流识别缓冲中所有条目记录的访问地址进行全相联比较。若命中，则根据地址比较的结果更新条目的状态，状态转换如图 3 所示。若不命中，则使用 LRU 算法淘汰缓冲中的一个条目，并将最新的访存信息登记到条目中。在图 3 中，“向前访问缺失”、“向后访问缺失”指访问一级 Cache 缺失且访问地址等于条目记录的 Cache 行地址加 1 或减 1；当条目记录的向前访问缺失或向后访问缺失的次数大于设定的流识别阈值时，条目进入向前访问预取（PFHFWD）或向后访问预取（PFHBWD）状态，即单位步长的访问流被成功识别。后续流水线上的访存请求符合流访问特征且访问“流”中新的 Cache 行地址，会触发预取引擎对一级、二级、三级 Cache 同时进行预取。

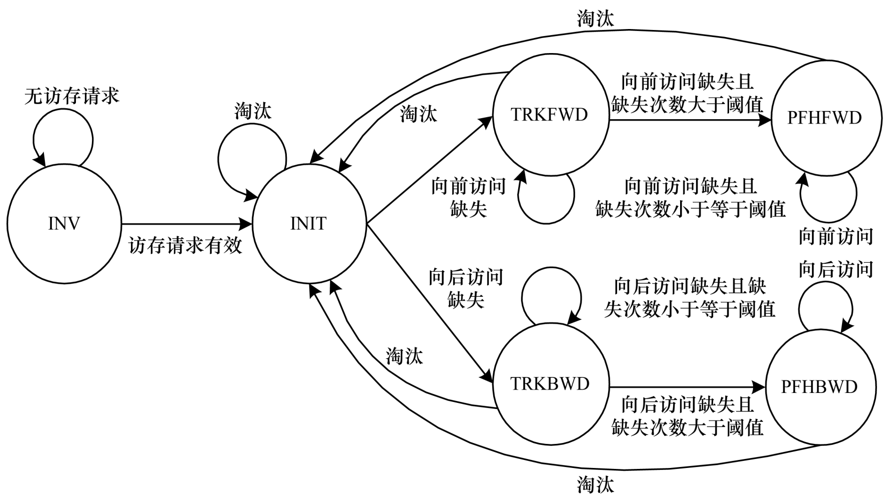

图 3　流缓冲中条目的状态转换

假设访问新的 Cache 行所需的周期数为 C,一级、二级、三级 Cache 的预取距离分别为 L_1, L_2, L_3。处理器访问 Cache 行 A 时,预取($A + L_1$)至一级 Cache,预取($A + L_2$)至二级 Cache,预取($A + L_3$)至三级 Cache。对多级 Cache 的预取过程如图 4 所示。

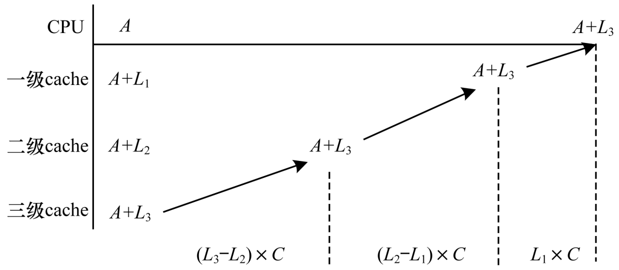

图 4 同时对三层 Cache 进行预取的过程

为了尽量保证处理器访存时,访问的数据已预取至一级数据 Cache,由图 4 可以看出:

(1)预取引擎发出预取($A + L_3$) 的请求至一级 Cache 与处理器访问($A + L_3$)的间隔为($L_1 \times C$)个时钟周期,在此期间内,预取数据必须从二级 Cache 装填至一级 Cache。

(2)预取引擎发出预取($A + L_3$)的请求至二级 Cache 与发出预取($A + L_3$)至一级 Cache 的请求间隔($L_2 - L_1$) $\times C$ 个周期,在此期间内,预取数据必须从三级 Cache 装填至二级 Cache。

(3)预取引擎发出预取($A + L_3$)至三级 Cache 的请求与发出预取($A + L_3$)至二级 Cache 的请求间隔($L_3 - L_2$) $\times C$ 个周期,在此期间内,预取数据必须从主存装填至三级 Cache。

由上述分析和表 1 中列出的各级 Cache 的访问延迟,参数 L_1, L_2, L_3 必须满足公式:

$$\begin{cases} L_1 \times C \geqslant 11 \\ (L_2 - L_1) \times C \geqslant 45 \\ (L_3 - L_2) \times C \geqslant 120 \end{cases}$$

在图 1 中,对于 SPEC CPU2000 测试集中的所有程序,流预取访问新的 Cache 行所需的周期数小于 200 的比例最低为 97.62% 。表 3、表 4 针对测试集中的所有程序,列出了访问新的 Cache 行所需周期数在 200 内的分布情况和对应的加权平均值(AVG)。假设流预取访问新的 Cache 行所需周期数在“0 ~ 10”,“10 ~ 40”,“40 ~ 120”,“120 ~ 200”各区间内所占的比例分别为 $R_{0\sim10}, R_{10\sim40}, R_{40\sim120}$ 以及 $R_{120\sim200}$,本文采用的加权平均值的计算方式如下:

$$AVG = R_{0\sim10} \times \frac{0+10}{2} + R_{10\sim40} \times \frac{10+40}{2} + R_{40\sim120} \times \frac{40+120}{2} + R_{120\sim200} \times \frac{120+200}{2}$$

表 3 SPEC CPU2000 整数程序在各区间所占比例

程序编号	周期区间				
	0 ~ 10	10 ~ 40	40 ~ 120	120 ~ 200	AVG
164	0.02	0.79	0.15	0.03	36.78
175	0.15	0.53	0.25	0.07	45.69
176	0.22	0.60	0.16	0.02	31.72
181	0.00	0.40	0.39	0.19	71.25
186	0.24	0.25	0.26	0.24	66.11
197	0.04	0.37	0.48	0.11	64.57
252	0.29	0.30	0.26	0.14	52.87
253	0.19	0.59	0.17	0.05	36.61
254	0.00	0.34	0.36	0.28	81.95
255	0.01	0.45	0.39	0.14	64.23
256	0.01	0.40	0.35	0.22	73.80
300	0.13	0.35	0.27	0.23	67.94
最小值	–	–	–	–	31.72

表 4 SPEC CPU2000 浮点程序在各区间所占比例

程序编号	周期区间				
	0 ~ 10	10 ~ 40	40 ~ 120	120 ~ 200	AVG
168	0.00	0.23	0.55	0.20	81.39
171	0.00	0.25	0.36	0.37	94.22
172	0.02	0.31	0.40	0.26	81.12
173	0.00	0.25	0.36	0.37	93.93
177	0.02	0.52	0.32	0.12	58.40
178	0.00	0.24	0.36	0.37	94.48
179	0.34	0.47	0.17	0.02	30.71
183	0.03	0.25	0.35	0.35	90.32
187	0.03	0.29	0.32	0.35	88.56
188	0.16	0.83	0.00	0.00	22.48
189	0.67	0.17	0.07	0.08	25.63
191	0.17	0.22	0.30	0.29	76.59
最小值	–	–	–	–	22.48

从表 3、表 4 中可以看出,对于 SPEC CPU2000 整数程序,流预取访问新的 Cache 行所需周期数加权平均值的最小值为 31.72,浮点程序为 22.48。当 C 的值取 22.48 时,参数 L_1, L_2, L_3 必须满足公式:

$$\begin{cases} L_1 \geqslant \left\lceil \frac{11}{22.48} \right\rceil = 1 \\ (L_2 - L_1) \geqslant \left\lceil \frac{45}{22.48} \right\rceil = 3 \\ (L_3 - L_2) \geqslant \left\lceil \frac{120}{22.48} \right\rceil = 6 \end{cases}$$

预取距离 L_1, L_2, L_3 的最小取值分别为 1,4,10。

4 实验结果与分析

4.1 实验环境

本文以一款国产申威处理器的逻辑设计为基础,使用硬件描述语言实现了基于流访问特征的多级 Cache 硬件预取。为了尽量保证性能测试结果的准确性,采用 Cadence 公司的 Palladium XP 系列硬

件仿真加速器[14]作为实验平台。实验以 SPEC CPU2000 作为基准测试程序集，并将测试程序在仿真加速器上的运行周期数作为性能评价的标准。实验的参数配置如表 5 所示。

表 5　实验参数配置

参数	配置
一级 Cache 预取距离	1
二级 Cache 预取距离	4
三级 Cache 预取距离	10
流识别缓冲条目数	16
流识别阈值	3

4.2　性能评测

本文以不使用硬件预取技术时，SPEC CPU2000 测试集中各程序在硬件仿真加速器上的运行周期数作为参照，对比了只对一级 Cache 进行预取单层预取（ONLY_DC）和同时对三层 Cache 进行预取多层预取（ALL_CACHE）对性能的提升情况。整数程序、浮点程序的比较结果分别如图 5、图 6 所示。

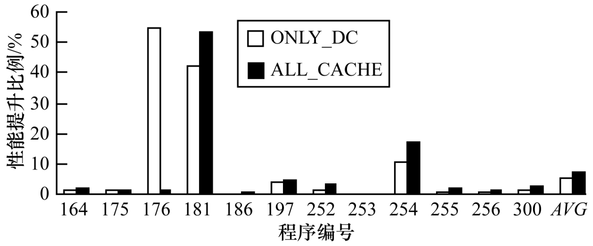

图 5　单层和多层预取对整数程序的性能提升

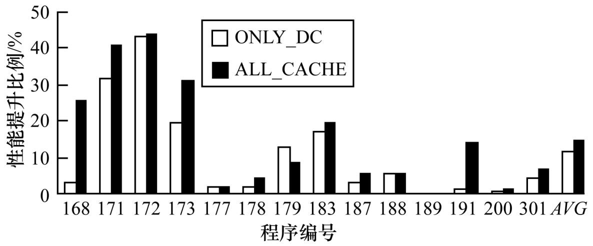

图 6　单层预取和多层预取对浮点程序的性能提升

从图 5、图 6 中可以看出，与只对一级 Cache 进行预取相比，同时对三层 Cache 进行预取可以将整数程序的性能平均提升 2.11%，最高提升 11.19%（181.mcf），浮点程序的性能平均提升 3.08%，最高提升 12.77%（191.fma3d）。

5　结束语

硬件数据预取技术可以降低 Cache 访问缺失造成长延迟问题对处理器性能的影响。本文结合现代处理器中各级 Cache 的容量和访问延迟差异，提出了同时对多级 Cache 进行预取的方法，并基于流访问特征的预取进行实现。基于硬件仿真加速器的实验结果表明，与仅对一级 Cache 预取相比，同时对多级 Cache 进行预取可以明显提升处理器的性能。本文方法有效解决了预取时机与 Cache 容量之间的矛盾，且该种方法并不局限于基于流访问特征的预取，对其他预取机制也同样适用。今后将考虑针对现代处理器多核、多线程结构的结构特征对本文方法进行优化。

参考文献

[1] Wulf W A, McKee S A. Hitting the Memory Wall: Implications of the Obvious[J]. ACM SIGARCH Computer Architecture News, 1995, 23(1): 20-24.

[2] Mowry T C, Lam M S, Gupta A. Design and Evaluation of a Compiler Algorithm for Prefetching[J]. ACM SIGPLAN Notices, 1992, 27(9): 62-73.

[3] Gindele J D. Buffer Block Prefetching Method[J]. IBM Technical Disclosure Bulletin, 1977, 20(2): 696-697.

[4] Jouppi N P. Improving Directed-mapped Cache Performance by the Addition of Small Fully-associative Cache and Prefetching Buffers[J]. ACM SIGARCH Computer Architecture News, 1990, 18(3): 364-373.

[5] Palacharls S, Kessler R E. Evaluating Stream Buffer as a Secondary Cache Replacement[J]. ACM SIGARCH Computer Architecture News, 1994, 22(2): 24-33.

[6] Iacobovici S, Spracklen L, Kadambi S, et al. Effective Stream-based and Eexecution-based Data Prefetching[C]// Proceedings of the 18th Annual International Conference on Supercomputing. New York, USA: ACM Press, 2004: 1-11.

[7] 靳　强，郭　阳，鲁健壮．一种步长自适应二级 Cache 预取机制[J]．计算机工程与应用，2011，47(29)：56-59.

[8] Intel Corporation. Intel 64 and IA-32 Architecture Optimization Reference Manual[EB/OL]. [2014-11-19]. http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf.

[9] Tendler J M, Dodson J S, Fields J S, et al. Power4 System Microarchitecture[J]. IBM Journal of Research and Development, 2002, 46(1): 5-25.

[10] Sinharoy B, Kalla R N, Tendler J M, et al. Power5 System Microarchitecture[J]. IBM Journal of Research and Development, 2005, 49(45): 505-521.

[11] Le H Q, Starke W J, Fields J S, et al. IBM Power6 Microarchitecture[J]. IBM Journal of Research and Development, 2007, 51(6): 639-662.

[12] 廖秋林，莫　玮，陈大为．SPEC CPU2000 性能测试程序分析及其应用[J]．国外电子测量技术，2006，25(6)：65-68.

[13] Henning J L. SPEC CPU2000: Measuring CPU Performance in the New Millennium[J]. Computer, 2000, 33(7): 28-35.

[14] Cadence Design Systems, Inc.. Cadence Palladium XPSeries[EB/OL]. [2014-10-19]. http://www.cadence.com/products/sd/palladium_xp_series/pages/default.aspx.

编辑　索书志