doi:10.11887/j.cn.201605010 http://journal.nudt.edu.cn

梯度学习的参数控制帮助线程预取模型*

裴颂文[1,2],张俊格[1],宁 静[1]
(1.上海理工大学 光电信息与计算机工程学院,上海 200093;
2.上海理工大学 上海市现代光学系统重点实验室,上海 200093)

摘 要:对于非规则访存的应用程序,当某个应用程序的访存开销大于计算开销时,传统帮助线程的访存开销会高于主线程的计算开销,从而导致帮助线程落后于主线程。于是提出一种改进的基于参数控制的帮助线程预取模型,该模型采用梯度下降算法对控制参数求解最优值,从而有效地控制帮助线程与主线程的访存任务量,使帮助线程领先于主线程。实验结果表明,基于参数选择的线程预取模型能获得1.1~1.5倍的系统性能加速比。

关键词:数据预取;帮助线程;多核系统;访存延迟;梯度下降
中图分类号:TN95 **文献标志码**:A **文章编号**:1001-2486(2016)05-059-05

Helper thread pre-fetching model based on learning gradients of control parameters

PEI Songwen[1,2], *ZHANG Junge*[1], *NING Jing*[1]
(1. School of Optical Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China;
2. Shanghai Key Laboratory of Modern Optical System, University of Shanghai for Science and Technology, Shanghai 200093, China)

Abstract: To the applications with irregular accessing memory, if the overhead of accessing memory for a given application is much greater than that of computation, it will make the helper thread lag behind the main thread. Hereby, an improved helper thread pre-fetching model by adding control parameters was proposed. The gradient descent algorithm is one of the most popular machine learning algorithms, which was adopted to determine the optimal control parameters. The amount of the memory access tasks was controlled by the control parameters effectively, which makes the helper thread be finished ahead of the main thread. The experiment results show that the speedup of system performance is achieved by 1.1 times to 1.5 times.

Key words: data pre-fetch; helper thread; multi-core system; memory latency; gradient descent

在微处理器的发展进入多核时代[1-2]之后,处理器和存储器之间的速度差距进一步拉大,存储墙[3]问题仍然是制约微处理器性能提升的一个重要瓶颈。数据预取技术[4]是缓解存储墙问题的重要手段之一,数据预取利用程序访存和计算的重叠,在处理器访问数据之前提前发出访存请求,隐藏因Cache缺失而引起的访存延迟。

传统的数据预取可分为硬件预取[5]和软件预取[6]两种。硬件预取在预取引擎的控制下,根据访存历史,对程序访存的模式进行识别和预测,通过硬件机制来预测可能发生的Cache失效,自动进行预取。但是,此种预取方式增加了硬件复杂性。软件预取是指由程序员或编译器在代码中插入预取指令,提前将数据取入Cache,从而避免在计算时由于数据缺失导致的执行暂停。

帮助线程预取技术[7]实质上是一种Leader/Follower结构,帮助线程是去除了原程序计算任务的"精简版本",它往往比主线程运行得快,因此帮助线程可提前于主线程发出访存请求,从而加速程序执行速度。帮助线程仅仅起到预取的作用,不修改主线程的体系结构状态,因此不会引起程序的错误执行。在理想的情况下,主线程需要某个数据的时候,帮助线程恰好能将需要的数据预取到末级缓存(Last-Level Cache,LLC)。但是,如果访存开销和计算开销差别较大的时候,帮助线程并不能每次领先主线程,导致预取的数据不能及时到达,造成Cache污染。根据不同的程序中访存开销和计算开销的规模,可将程序划分为

* **收稿日期**:2015-11-16
基金项目:上海市自然科学基金资助项目(15ZR1428600);计算机体系结构国家重点实验室开放资助项目(CARCH201206);上海市浦江人才计划资助项目(16PJ1407600)
作者简介:裴颂文(1981—),男,湖南邵东人,副教授,博士,硕士生导师,Email:swpei@usst.edu.cn

以下三种类别。设程序的访存时间为 T_m,计算时间为 T_c。

1)计算开销与访存开销大小相当,即 $T_m \approx T_c$。此时帮助线程能很好地发挥作用。

2)计算开销大于访存开销,即 $T_c > T_m$。此时要控制好帮助线程的预取时机,防止预取时机过早,从而导致真正使用的时候数据已被替换出去。

3)计算开销小于访存开销,即 $T_c < T_m$。此时主线程计算开销小,帮助线程访存开销大,主线程的数据没有被帮助线程预取到,主线程可能要进行多次同步操作。

对于非规则数据访存的应用程序,其数据结构通常使用图、树或者链表等组成。此类应用程序的访存行为呈现非规则性,访存模式难以在静态编译阶段进行准确的预测[8]。由于其可利用的局部性受到限制,使得传统的软件和硬件预取方法失效,其访存模式只能通过执行代码本身来进行预测。由于非规则访存密集型程序往往带来大量的访存开销,并且远远大于计算开销,因此本文着重针对第三种类别,提出参数控制的帮助线程预取方法。帮助线程负责访存任务,主线程负责计算任务。帮助线程提前将主线程所需的数据预取到 LLC,从而达到隐藏访存延迟的目的。

1 相关工作

帮助线程技术可以通过硬件与软件的方法实现[9]。硬件方法通过指令窗口动态生成帮助线程,硬件复杂度较高。软件方法是对程序的源代码进行剖析,由编译器显式插入预取线程代码,易于实现。

Kim 等[10]利用 Unravel 切片工具和斯坦福大学 SUIF 编译框架在源代码级完成了帮助线程的自动构造。利用预取转换(Prefetch Conversion, PC)操作来进行主线程和帮助线程之间的同步,通过设置主线程与帮助线程计数器的方式来控制帮助线程的执行速度。只有当两个线程计数器之差大于一个特定的阈值 PD(预取距离)时,帮助线程才继续运行。Song 等[11]在 SUNSPARC 平台上基于编译实现了帮助线程的构造方法,该方法通过判断帮助线程的收益来进行构造。Ou 等[12]提出的基于线程的预取方法,通过在处理器上添加动态预取线程构造逻辑和控制逻辑,对程序的访存特点进行分析,并从主线程的执行行踪中提取数据预取线程,使用空闲的线程和主线程并行执行。Yu 等[13]提出了一种线程感知的自适应的数据预取方法,根据线程动态反馈信息将线程进行分类,从硬件层面控制线程的竞争,但是需要物理模块的支持。

以上面向帮助线程预取的研究大多集中于主线程和帮助线程的构造与同步机制。但是,对于实际的应用程序,在计算开销很小的情况下,帮助线程不一定总快于主线程,此时就会频繁产生同步操作,导致程序性能下降。因此,如何能够合理分配一定比例的访存任务由主线程完成,使得帮助线程与主线程协同工作,从而有效提高系统访存和计算性能,是本文研究的重点。

2 参数控制的预取模型

通过以上分析,帮助线程的访存开销分为两种:①对于计算密集型的应用程序,帮助线程承担全部的访存任务;②对于访存密集型的应用程序,帮助线程承担部分的访存任务。如果让帮助线程取得较好的性能,必须根据程序不同的访存开销与计算开销来调整帮助线程预取数据量的大小。帮助线程应从热点程序入口处开始跳过 K 个数据块之后才开始推送 P 个数据块,从而提高帮助线程预取数据的有效性与及时性。在预取的时候,一方面要保证帮助线程能够及时地预取主线程所需要的数据,另一方面要保证帮助线程不会落后或超前于主线程太长的距离,从而替换掉主线程所需的有用数据,造成多核平台的最后一级缓存污染。

2.1 预取参数的定义

采用 Zhang 等[14]提出的 KPB(skip-push-block)参数,在考虑原程序计算访存工作量的前提下,通过动态调整 K,P,B 三个参数值,使得帮助线程的性能达到最优。将热点模块按照循环数分成等长的 Block,一次循环所需数据称作一个数据块。

1)K 即 skip,表示帮助线程跳过多少个数据块,即主线程负责 K 个数据块的访存,其他的访存任务由帮助线程来完成,此参数主要用于控制帮助线程预取的触发时机。若程序的计算开销远远大于仿存量,此时 $K=0$,与传统的帮助线程预取机制一样,帮助线程承担全部的访存工作。

2)P 即 push,表示帮助线程给主线程推送多少个数据块,即帮助线程预取的数据量。此参数用于控制帮助线程预取工作量的大小。

3)B 即 block,表示帮助线程与主线程多长时间同步一次。一般情况下 $B=K+P$。此参数用于控制帮助线程与主线程的同步频次,采用文

献[10]所述的线程同步机制。

目前参数的选取大多都是通过枚举实验来获取,设 $R_P = P/(K+P)$,其中 R_p 为预取率,$0 < R_p < 1$,选取符合 R_p 范围 K,P 进行多次试验,最后选取实验中平均每条指令的时钟周期数(Cycles Per Instruction, CPI)最小的 K,P 参数组合作为最优值。

2.2　参数控制预取模型

基于梯度学习的参数控制预取模型主要由两部分组成,即热点分析和代价函数的构造。预取算法的基本流程如图 1 所示。

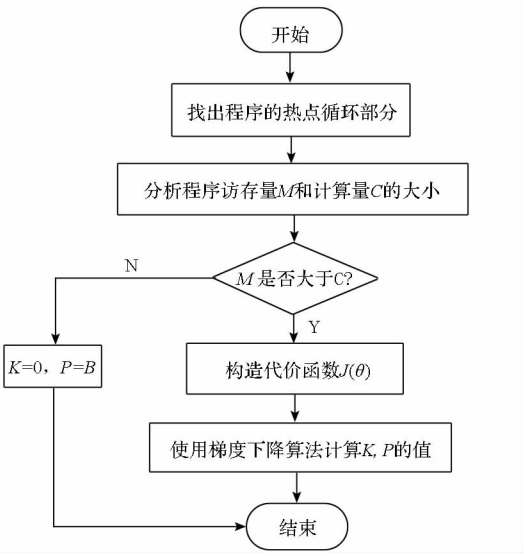

图 1　参数控制的预取模型流程图

Fig. 1　Flow chart of pre-fetching model based on control parameters

2.2.1　确定程序的热点部分

主要面向的测试对象为非规则访存应用程序,其数据结构一般是非线性的链式结构,相对规则访存程序的访存时间局部性和空间局部性较差。在运行过程中,非规则访存程序产生访问 Cache 缺失的概率比较高,对测试程序的总体执行性能影响较大。首先使用 Intel 性能分析工具 Vtune[15] 对此类程序进行离线 Profiling,收集 CPU 的时钟周期和共享 Cache 的缺失信息,然后找出引起 Cache 缺失的长延迟访存指令,确定要进行预取的热点循环部分。热点循环要满足以下两个基本条件:

1)热点循环中包含长延迟的间址访存指令;

2)存在 $\varepsilon 1$, $\varepsilon 2$ 满足 $\frac{Cycle(f(p))}{Cycle(p)} \geqslant \varepsilon 1$, $\frac{LLC\_miss(f(p))}{LLC\_miss(p)} \geqslant \varepsilon 2$, $0 < \varepsilon 1 < 1$, $0 < \varepsilon_2 < 1$。其中:$Cycle(f(p))$ 表示程序 p 的热点模块 $f(p)$ 的时钟周期数,$Cycle(p)$ 表示程序 p 的时钟周期数;$LLC\_miss(f(p))$ 表示程序 p 的热点模块 $f(p)$ 的 Cache 缺失率,$LLC\_miss(p)$ 表示程序 p 的 Cache 缺失率。根据经验值 $\varepsilon 1 > 0.5$, $\varepsilon 2 > 0.5$ 时,预取效果比较明显。

2.2.2　构造代价函数

通过 Vtune 工具,分析热点程序的访存任务量 M(访存所消耗的时间)与计算任务量 C(计算所消耗的时间)的大小。为确保程序性能最优,将满足的条件设为代价函数,记作 $J(\theta)$。根据预取模型,主线程负责的任务是:

1) K 个数据块的访存与计算;

2) P 个数据块的计算。

可记作 $K(T_c + T_m) + PT_c$。其中:T_m 为单次循环的访存时间,T_c 为单次循环的计算时间。

帮助线程负责的任务为:P 个数据块的预取,可记做 PT_m。

理想情况下,帮助线程与主线程完全并行,此时程序性能达到最优,即

$$|K(T_c + T_m) + P(T_c - T_m)| \tag{1}$$

的绝对值最小。

假设给主线程分配的访存任务为 m_0,根据经验可知,随着 K 的增加,m_0 也增加。K,m_0 的关系可设为:

$$K = \theta_0 \cdot m_0 \tag{2}$$

假设帮助线程分配的访存任务为 m_1,同样,根据经验可知,随着 P 的增加,m_1 也增加。P,m_1 关系可设为:

$$P = \theta_1 \cdot m_1 \tag{3}$$

将式(2)、式(3)代入式(1)得:

$$\theta_0 \cdot m_0(T_c + T_m) + \theta_1 \cdot m_1(T_c - T_m)$$

根据上述推断,可知代价函数为:

$$J(\theta) = \frac{1}{2}[\theta_0 \cdot m_0(T_c + T_m) + \theta_1 \cdot m_1(T_c - T_m)]^2 \tag{4}$$

2.2.3　计算最优的 K,P

梯度学习作为一种求解最优参数的迭代算法,广泛应用于机器学习各式 model 参数的求解中。梯度下降算法是一种迭代方法,利用负梯度方向来决定每次迭代的新的搜索方向,使得每次迭代能使待优化的目标函数逐步减小。因此,选择梯度下降算法进行最优值的求解,通过选择不同的 m_0,m_1 的样本值,可以训练出满足代价函数 $J(\theta)$ 最小的 θ_0,θ_1。

通过 $m_0 + m_1 = M$,即

$$K/\theta_0 + P/\theta_1 = M \tag{5}$$

又有

$$K + P = B \tag{6}$$

通过式(5)、式(6),可以得出近似最优解 K, P 的值。由于求解的 K,P 可能为小数,因此可以在近似最优解附近$[k-\sigma_1, k+\sigma_2]$,选三组整数数据进行测试,通过运行程序,使用 Vtune 分析实验结果,得到 CPI 最小的 K,P 即为最优解。

2.2.4 构造帮助线程

利用 Vtune 确定热点循环,然后构造有效的、轻量级的帮助线程。通过切片工具从热点循环中提取不包含计算部分的代码,编译器根据 profile 文件信息将要预取的访存指令标记为关键指令,将计算指令标记为非关键指令。最终,将关键指令抽取出来,形成帮助线程的代码块。帮助线程与主线程之间通过共享变量的方式进行同步和通信。帮助线程每跳过 K 个数据块,预取 P 个数据块后,就要和主线程同步一次。

3 实验分析

实验选取的测试程序为 Olden Benchmark 中用于科学计算的测试程序 EM3D、MST,SPEC CPU 2006 中的 MCF 进行帮助线程预取性能的评估。处理器是 Intel® Core™ 2 Q6600 四核处理器,该处理器共有 8 MB 二级高速缓存,每对核共享 4 MB二级高速缓存。通过 Vtune 的分析,选取的热点模块以及输入集见表 1,分别为 EM3D 中的 Fill_from_field,MST 中的 Hashlookup,MCF 中的 Refresh_potential。

表 1 Benchmark 参数配置表

Tab. 1 Benchmark parameter setting

Benchmark	热点函数	输入
EM3D	Fill_from_fields	400 000 nodes, arity 128
MCF	Refresh_potential	Ref
MST	Hashlookup	10 000 nodes

如图 2 所示,EM3D,MST 和 MCF 测试程序采用传统帮助线程(帮助线程负责全部的访存任务)、参数枚举法和基于梯度学习的参数控制方法(参数学习法)相对于串行执行(不使用帮助线程的源程序)时的性能加速比,其中参数学习法获得了 1.1 ~ 1.5 倍的最高加速比。MST 的 Hashlookup 模块属于访存密集型程序,使用传统的帮助线程方法与原串行程序相比加速比反而降低了 4.8%;使用参数学习方法,性能提升了近 50%。EM3D 的 Fill_from_fields 模块属于计算量较大的程序,使用参数学习方法与传统帮助线程方法获得的加速比相当,仅提高了 4.9%。由于参数枚举法取决于经验与启发式实验,枚举粒度的大小直接影响到结果的准确性。粒度过小,需要进行大量的重复试验;粒度过大,可能错过最优值。因此,参数枚举法并不总能得到最优解。参数学习法不依赖于经验,而是通过机器学习的方法获取最优值,比参数枚举法效率更高。

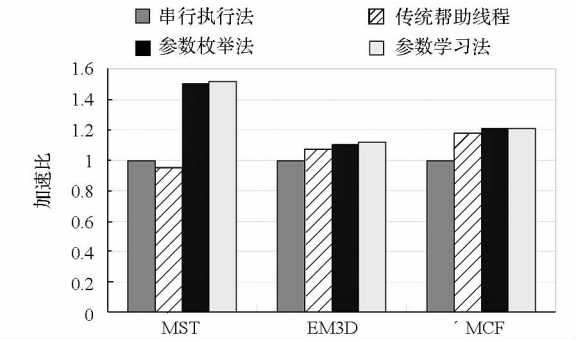

图 2 性能加速比

Fig. 2 Performance speedup

图 3 给出了测试程序在使用传统的帮助线程、参数枚举法和参数学习法情况下各自 Cache 缺失率的归一化相对值。其中,MST,EM3D,MCF 的热点程序的 Cache 缺失率相对于采用传统帮助线程的情况下分别减少了 12%,10%,27%。MST,EM3D 相对于参数枚举法分别减少了 2.5%,1.7%。因此,通过机器学习的梯度下降算法取得 K,P 的最优值比参数枚举法效率更高。

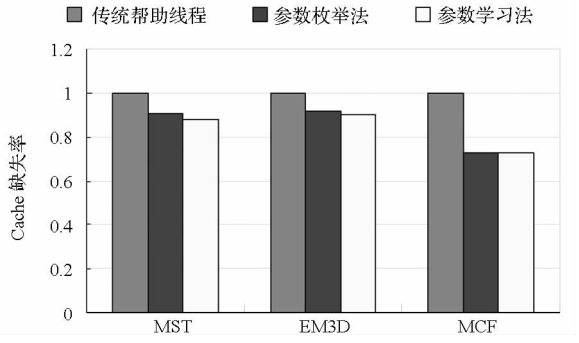

图 3 Cache 缺失率

Fig. 3 Cache missing rate

预取的准确率等于帮助线程有效预取的次数与其发出的全部预取次数的比例。对比串行执行的 Cache 缺失率与采用参数学习的帮助线程预取技术后的 Cache 缺失率,评估预取的准确率与覆盖率。如图 4 所示,相对于原串行执行的方法,基于参数学习的帮助线程预取算法对数据预取的效果是明显的,降低了数据 Cache 的缺失率。

由于活动计算核数量的增加以及对资源的竞争,采用帮助线程的程序执行相比于串行程序执行,将会在一定程度上增加功耗。如果帮助线程的

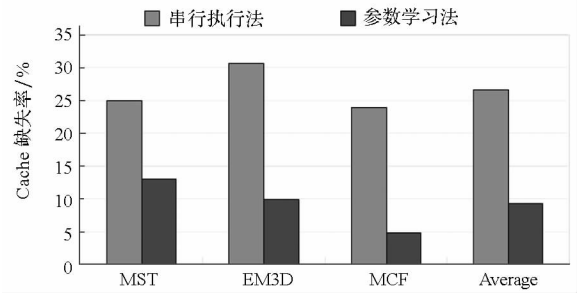

图 4　Cache 缺失率对比

Fig. 4　Comparison of Cache missing rate

收益大于额外增加的功耗,则体现了帮助线程的有效性。帮助线程额外增加的功耗表示相对于串行程序执行功耗[16]的比例。帮助线程收益表示相对于串行程序执行时间所减少的比例。如图 5 所示,帮助线程的平均收益大于平均功耗。其中,MST、MCF 的收益均大于功耗,因此帮助线程能有效提升 MST、MCF 的执行性能。因为 EM3D 属于计算量较大的程序,访存量较小,帮助线程反而带来了很大的同步开销,不足以弥补帮助线程带来的收益,所以,帮助线程对提高 EM3D 执行性能有限。

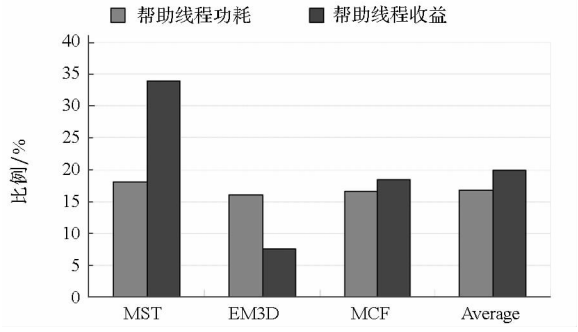

图 5　帮助线程功耗和收益比

Fig. 5　Comparison between energy overhead ratio and performance gain of helper thread

4　结论

通过分析传统的帮助线程不能有效地控制预取实时性和覆盖率缺陷,以及对访存密集型的程序预取效率低的劣势,提出了一种基于梯度学习的参数控制帮助线程预取模型。通过采用机器学习的梯度下降算法确定 K,P 的值,根据 K,P 的值选择性地预取部分数据,使得帮助线程与主线程的工作量相对均衡,从而使程序的执行性能达到最优。

由于帮助线程与主线程同时访存,可能会引起带宽的竞争。因此,下一步的工作将考虑帮助线程对带宽的影响,具体分析程序的访存地址,适当增加预取步长,将预取相邻地址的预取指令进行合并,以减少预取次数,从而可以降低带宽的竞争,提高执行性能,降低额外功耗。

参考文献(References)

[1] Pei S W, Kim M S, Gaudiot J L. Extending amdahl's law for heterogeneous multicore processor with consideration of the overhead of data preparation[J]. IEEE Embedded Systems Letters, 2016, 8(1): 26 – 29.

[2] 裴颂文, 吴小东, 唐作其, 等. 异构千核处理器系统的统一内存地址空间访问方法[J]. 国防科技大学学报, 2015(1): 28 – 33.
PEI Songwen, WU Xiaodong, TANG Zuoqi, et al. An approach to accessing unified memory address space of heterogeneous kilo-cores system [J]. Journal of National University of Defense Technology, 2015(1): 28 – 33. (in Chinese)

[3] Wilkes M V. The memory wall and the CMOS end-point[J]. ACM Sigarch Computer Architecture News, 1995, 23(4): 4 – 6.

[4] Vanderwiel S P, Lilja D J. Data prefetch mechanisms[J]. ACM Computing Surveys, 2000, 32(2): 174 – 199.

[5] Ganusov I, Burtscher M. Future execution: a hardware prefetching technique for chip multiprocessors [C]// Proceedings of Parallel Architectures and Compilation Techniques Conference, 2005: 350 – 360.

[6] Dudás, Juhász S, Schrádi T. Software controlled adaptive pre-execution for data prefetching[J]. International Journal of Parallel Programming, 2012, 40(4): 381 – 396.

[7] Lee J, Jung C, Lim D, et al. Prefetching with helper threads for loosely coupled multiprocessor systems [J]. IEEE Transactions on Parallel & Distributed Systems, 2009, 20(9): 1309 – 1324.

[8] Huang Y, Tang J, Gu Z M, et al. The performance optimization of threaded prefetching for linked data structures[J]. International Journal of Parallel Programming, 2011, 40(2): 141 – 163.

[9] 张建勋, 古志民. 帮助线程预取技术研究综述[J]. 计算机科学, 2013, 40(7): 19 – 23.
ZHANG Jianxun, GU Zhimin. Survey of helper thread prefetching[J]. Computer Science, 2013, 40(7): 19 – 23. (in Chinese)

[10] Kim D, Yeung D. A study of source-level compiler algorithms for automatic construction of pre-execution code[J]. ACM Transactions on Computer Systems, 2004, 22(3): 326 – 379.

[11] Song Y, Kalogeropulos S, Tirumalai P. Design and implementation of a compiler framework for helper threading on multi-core processors[C]//Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2005: 99 – 109.

[12] Ou G D, Zhang M X. Thread-based data prefetching[J]. Computer Engineering & Science, 2008, 30(1): 119 – 122.

[13] Yu J Y, Liu P. A thread-aware adaptive data prefetcher[C]// Proceedings of Computer Design, Seoul, 2014: 278 – 285.

[14] Zhang J X, Gu Z M, Huang Y, et al. Helper thread prefetching control framework on chip multi-processor[J]. International Journal of Parallel Programming, 2013, 43(2): 180 – 202.

[15] Intel VTune performance analyzer for linux [EB/OL]. [2012 – 12 – 10]. http://www.intel.com/support/performacetools/vtune/linux.

[16] Singh K, Bhadauria M, Mckee S A. Prediction-based power estimation and scheduling for CMPs [C]//Proceedings of International Conference on Supercomputing, 2009: 501 – 502.