山东农业大学学报(自然科学版),2017,48(1):93-99 VOL.48 NO.1 2017
Journal of Shandong Agricultural University (Natural Science Edition) doi:10.3969/j.issn.1000-2324.2017.01.019

面向计算功耗效率的超线程调度研究

林何磊,冯国富*,冀续烨,陈 明
上海海洋大学 信息学院, 农业部渔业信息重点实验室, 上海 201306

摘 要: 相对于传统计算侧重关注计算性能及并行可扩展性而忽视功耗因素，本文分析 Hyper threading 服务器的功耗特性，基于超线程计算单元拓扑感知，引入面向能量效率的超线程调度策略，利用线程亲和性改变线程计算单元分配顺序以提高系统计算与能量效率，克服操作系统默认线程调度策略单一、呆板的缺陷。实验结果表明，所提方案可在应用层、运行时支持层和操作系统层三层中灵活配置，以影响不同范围内的计算应用，在提高能量效率的同时，从计算单元利用角度提高计算资源利用率。

关键词: 超线程; 能量效率; 计算单元调度; 计算单元拓扑感知
中图法分类号: TP332 **文献标识码**: A **文章编号**: 1000-2324(2017)01-0093-07

Study on Hyper Threading Scheduling towards Power Efficiency of Computing

LIN He-lei, FENG Guo-fu*, JI Xu-ye, CHEN Ming
College of Information Technology, Key Laboratory of Fisheries Information Ministry of Agriculture/Shanghai Ocean University, Shanghai 201306, *China*

Abstract: As traditional computing focuses on performance and parallel scalability while ignoring the power consumption factor, based on Computing unit topology awareness of Hyper Threading, this paper analyzes the power consumption of Hyper threading server, and introduces a Hyper Threading Scheduling Strategy for energy efficiency. By setting thread's processor affinity to change the allocation order of computing units, the proposed method improves the system computing and energy efficiency, and overcomes the shortage of traditional OS whose schedule strategy is too monotonous for Hyper Threading. The experimental result shows that the proposed scheme can be deployed flexibly in three layers: application layer, runtime library layer and operating system kernel layer to affect computing applications in different scope. While improving energy efficiency, the utilization of computing resources are improved too.

Keywords: Hyper threading; energy efficiency; computing unit scheduling; computing unit topology awareness

SMT 计算单元与传统处理器单元存在区别，但 OS 通常不能有效感知 SMT 计算单元与传统计算单元的不同，从而造成 SMT 的技术优势不能得以发挥。以上特点使得 SMT 技术虽从 95 年左右就已出现，但一直没有得到很好的推广。Intel 早在 2000 年的 P4 Netburst 微架构就引入 Hyper Threading，但在 2006 年 Core 微架构中又放弃该技术。当前低碳节能已成为全球最受关注的话题之一，并亦成为国内外 IT 领域的研究热点，计算结点的功耗不仅直接影响计算中心、数据中心的运营成本，还进一步间接影响用于散热方面的开销及对设备器件寿命的影响[1]。SMT 技术由于在资源、功耗效率方面的优点，再次受到研究界的关注，Intel 自 2008 年的 Nehalem 微架构又重新启用了 Hyper Threading 技术，但 Hyper Threading 不能被 OS 有效感知等问题仍然制约该技术的推广与普及。

1 Hyper Threading 线程调度中存在的问题

1.1 基于Hyper Threading 的计算系统

目前基于 Hyper Threading 的高性能服务器通常采用 NUMA 架构。如图 1 所示，在一个典型的双路 Intel Xeon 单结点服务器中，配置二路物理处理器，每路物理处理器中有 4 个处理核心 Cx（x=0,1,2,3），每个核心含两个 Hyper Threading 线程单元 Tx0,Tx1（x=0,1,2,3）。

虽然每个处理器核上的 2 个 Hyper Threading 线程单元是对等的，但第一个获得计算任务的线程单元可以独享处理器核心上全部执行资源直到另一线程单元上有任务执行。从 OS 线程调度角度而

收稿日期: 2016-11-03 **修回日期**: 2016-12-18
基金项目: 上海市科技创新行动计划项目:小龙虾生态智能化设施养殖关键技术研究与应用(16391902902);江苏省国家长江珍稀鱼类工程技术研究中心培育点(BM2013012)
作者简介: 林何磊(1992-),男,硕士,主研方向:高性能计算机系统结构. E-mail:heleilin@hotmail.com
***通讯作者**: Author for correspondence. E-mail:jt_f@163.com

数字优先出版:2017-02-23 http://www.cnki.net

言，由于分配任务的先后顺序及对计算资源的独享与共享，其二者又是不完全对等的。

为表述方便，在不至引起混乱的前提下，本文将处理器核心上独占执行资源、第一个执行任务的 Hyper Threading 线程称为全资源线程单元，而其后在相同计算资源上启动，并与之前独占资源的线程共享执行资源的线程单元称为共享线程单元。此时先前的全资源线程单元因为要和后者共享计算资源，自动演化为共享线程单元，也即共享线程单元是成对出现的。当计算任务申请的计算单元为奇数时，至少存在一个全资源线程单元。在对计算效率和成本进行分析比较时，以全资源线程单元为单位更具有可比性。

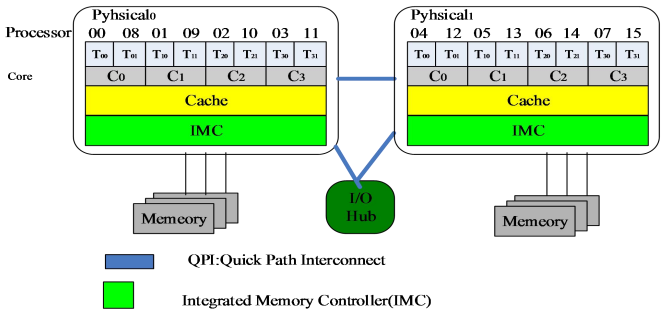

图 1 基于Hyper Threading 的计算系统

Fig.1 The computing system on Hyper Threading

传统 OS（如 Linux）在对 Hyper Threading 处理器线程单元进行管理时，通常是优先分配系统中的全资源线程单元，然后再分配共享线程单元。如本文实验表明，一台双路 Intel Xeon L5520 服务器基于 Linux 3.6.11 内核会按表 1 所示顺序来为计算任务分配计算资源。表 1 以“(Processor,Core,Thread), alloc_seq”的形式来标识一次分配中计算单元所处的“物理处理器号，处理器核心号与线程单元号及 Linux 内核分配计算资源的顺序”；黑色实线箭头指示全资源线程单元的分配顺序，虚线箭头指示共享线程单元的分配顺序。

由表 1 可以看出 Linux 计算单元分配策略首先以全资源线程单元来满足计算需求，当全资源线程单元不能满足需求时，才分配共享线程单元，从而使计算请求得到很好、很快地响应。这种分配方案对于密集事务型操作会获得比较好的用户响应和系统吞吐量。但如表 1 所示，这种分配策略通过快速覆盖全资源线程单元的来提高性能，却忽视了对共享线程单元的利用，没有充分发挥 SMT 技术的特点。

表 1 Linux 分配 SMT 计算单元的顺序

Table 1 The allocation sequence of Linux kernel for SMT units

			物理核心	Core 0	Core 1	Core 2	Core 3
物理处理器	Processor 0	线程单元	Thread 0	◎	◎	◎	◎
				（0,0,0）,1	（0,1,0）,2	（0,2,0）,3	（0,3,0）,4
			Thread 1	◎	◎	◎	◎
				（0,0,1）,9	（0,1,1）,10	（0,2,1）,11	（0,3,1）,12
	Processor 1		Thread 0	◎	◎	◎	◎
				（1,0,0）,5	（1,1,0）,6	（1,2,0）,7	（1,3,0）,8
			Thread 1	◎	◎	◎	◎
				（1,0,1）,13	（1,1,1）,14	（1,2,1）,15	（1,3,1）,16

1.2 OS 线程单元分配策略下的并行性能

由于受并行可扩展性的影响，并行计算中投入计算单元的数量通常有一个上限。超过该上限后，再投入新的计算资源，计算资源虽然增多，但性能的提高却不再明显。

依据 Amdahl 定律：令 S 为某一计算中难以并行化执行的部分，P 为可并行的部分(S+P=1)，那么投入 N 个计算单元最大可能的加速比 SN 为：

$$SN=\frac{S+P}{S+\frac{P}{N}}=\frac{1}{S+\frac{P}{N}}=\frac{N}{(N-1)S+1}\qquad\text{（公式-1）}$$

用 $\frac{SN}{N}$ 代表计算的并行效率，以 S=0.1 为例，从并行效率考虑，当 $\frac{SN}{N}<\frac{1}{2}$ 时，再投入新的计算单元加速已不明显（即：N>1/S+1）。也即，在上述假设下 N>11 后加速效果不再明显。目前随着制造工艺的发展，单个处理器的核心数越来越多，SMP 服务器处理器路数也不断提高。如 HP DK388P 采用 2 路 XeonE5-2667，每处理器 6 核心、12 线程；浪潮正睿 6540S2，采用四路 Xeon E7-8850 处理器，每处理器 10 核心，20 线程。如果按照 Linux 默认的分配策略，随着计算结点内计算单元增多，受限于可扩展性，大量计算单元要么没有投入计算的机会，要么投入后对计算的贡献微乎其微。在 Hyper Threading 架构中，因为 OS 要分配将全资源线程单元分配完后再分配共享线程单元，而共享线程单元投放后，要和另一个单元共享计算资源，共享线程单元的计算能力只相当于全资源线程单元的σ倍（$0<\sigma<1/2$）（经典文献中认为 Hyper Threading 可以在典型环境下提高 30%的计算性能），这使得 Hyper Threading 单元的贡献更不明显。

表 2 Phoenix2.0 下 Word counter 执行时间

Table 2 Phoenix 2.0 parallel computing for Word counter (s)

核心数	3200 M 案例 Case			6400 M 案例 Case		
Cores	执行时间（s）	SN	SN/N	执行时间(s)	SN	SN/N
1	40.98	1	1	76.31	1	1
2	21.61	1.89	0.94	39.71	1.92	0.96
3	14.32	2.86	0.95	28.23	2.70	0.90
4	11.75	3.48	0.87	22.62	3.37	0.84
5	9.81	4.17	0.83	18.26	4.18	0.86
6	8.37	4.89	0.81	15.89	4.80	0.80
7	7.73	5.30	0.75	13.86	5.50	0.79
8	7.11	5.76	0.72	12.06	6.33	0.79
9	6.32	6.475	0.71	10.85	7.03	0.78
10	6.06	6.762	0.67	11.16	6.84	0.68
11	5.74	7.13	0.64	9.97	7.65	0.69
12	6.34	6.45	0.53	10.07	7.58	0.63

表 2 为在 HP DK388P，基于 Linux 2.6.32-131.0.15.el6.x86_64 内核运行共享存储模式 Mapreduce 实现 Phoenix 2.0[2]中 word counter 案例的并行执行时间（数据集大小分别为 3200 M、6400 M）。由于 Linux 是先分配全资源线程单元，而 HP DK388P 全资源线程单元只有 12 个。表 2 显示，当申请计算单元数达到全资源线程单元数上限 12 后（SN/N 分别达到 0.54 和 0.63），加速比不再明显上升。当 12 个全资源线程单元分配完后，Linux 才会再投入共享线程单元，然而此时投入共享线程单元不仅会增大系统功耗，同时对于性能的提升作用不大，这也是多数高性能应用选择关闭 SMT 功能的原因(本文按参考文献[2,3]对实验案例进行了优化，不优化情况下并行执行效果会更差)。

综上所述，操作系统单一、僵化的 Hyper Threading 计算单元分配策略过晚地投放共享线程单元，使得 Hyper Threading 性能得不到有效发挥，造成计算资源、功耗与调度机会的浪费。另外对于 OS 而言，全资源线程单元与共享线程单元统一被视为独立的计算单元，忽视了全资源线程单元和共享线程单元二者计算能力不对等的特点，使基于 SMT 的共享存储并行计算易引入负载均衡问题。因此研究更灵活的 Hyper Threading 计算单元分配策略是绿色、高效计算的内丰需求，通过改变并行计算分配线程单元策略，让 Hyper Threading 共享线程单元在合适的时机“贡献”计算能力，能更好利用 SMT 机制的 IPL、TPL 双重并行性、充分挖掘 SMT 在功耗与成本上的优势，便于用户寻找性能与功耗的平衡点。

2　基于超线程拓扑感知的Hyper Threading 线程调度策略

2.1　OS 线程单元功耗/成本分析

OS 僵化的线程单元分配策略不能充分发挥 SMT ILP 和 TLP 的技术特点。SMT 的优势在于使用较少的附加资源（包括功耗）来挖掘执行单元的利用率。

设 Hyper Threading 计算结点有 N_p 个物理处理器，每个物理处理器上有 N_c 个核，每核 2 个 SMT

线程单元。假设优先在已有负载的处理器上分配全资源线程单元。

设结点(Node)空载功率为 P_n，每投入一个物理处理器（Socket）功率增加值为 P_s，投入一个全资源线程单元(Core)功率增加值为 P_c，投入一个物理 $P_c = P_{n0} + \left\lceil \frac{c}{N_c} \right\rceil P_{s0} + CP_{c0}$

线程(Thread)单元功率增加值为 P_t。

则若不启用 Hyper Threading，单计算结点上启用线程数为 T，计算结点功率为：

如果启用 Hyper Threading，单计算结点启用线程数为 T（为简单记，设 T 为偶数）计算功率为：

$$P_T = P_{n0} + \left\lceil \frac{T}{2 * NC} \right\rceil * P_{s0} + \frac{1}{2} * T * P_{c0} + T * P_t$$

表 3 超线程单元功率（单位：W）

Table 3 The powers of hyper threading units (W)

线程单元数 Cells	0	1	2	3	4	5	6	7	8	9	10	11	12
功率1	76.18	115.21	122.4	130.17	141.09	155.56	166.3	174.33	184.18	187.47	189.8	192.35	194
功率2	77.25	110.94	122.32	130.34	141.4	156.09	166.78	173.85	184.46	187.62	190.11	192.34	194.18
功率3	75.37	111.45	122.83	130.34	141.46	155.99	166.62	173.94	184.59	187.73	190.15	192.53	194.53
功率4	76.96	111.25	122.94	130.62	141.47	156.12	167.23	174.11	184.78	187.76	190.11	192.13	194.54
功率5	75.88	111.15	122.84	130.47	141.36	155.89	167.19	174.6	184.36	187.52	190.09	192.46	194.44
功率6	74.99	110.23	122.67	130.42	141.47	157.24	166.85	174.65	184.83	187.88	190.29	192.4	194.82
均值	76.105	111.71	122.67	130.39	141.38	156.15	166.83	174.25	184.53	187.66	190.09	192.37	194.42
增加值		35.61	10.96	7.73	10.98	14.77	10.68	7.42	10.29	3.13	2.43	2.28	2.05

表 3 列出了在 Dell C6100 单结点服务器（Linux 3.6.11）上测得的不同个数计算单元满负荷情况下计算结点功率的变化情况。测量方法：使用 Linux 默认分配计算单元策略，当系统负荷与功率稳定后对系统功率值连续采样 8 s，每秒采样 1 次，去掉最大值与最小值，再求平均值。

从表 3 得出 Dell C6100 单结点服务器：

①P_n=76.11；

②当因负载需求使第一个物理 CPU 中第一个计算单元满载时（启动第一个物理 CPU），系统功率增加 35.61 W;

③使第二个物理 CPU 中第一个计算单元满载时（启动第二个物理 CPU，第 5 个核心投入计算时），系统功率增加约 14.77 W，为简化计算我们将 P_s 取②③的平均值 25.19 W;

④在物理 CPU 上启动 1 个全资源线程单元功率平均增加 P_c=9.68 W；

⑤而使一个共享线程单元满载功率平均增加 P_t=2.61 W。

同样方法测得，在 HP DK388P 采用 2 路 Xeon E5-2667 单结点服务器（Linux 2.6.32）测得：

①P_n=106.41；

②当因负载需求使第一个物理 CPU 中第一个计算单元满载时（启动第一个物理 CPU），系统功率增加 43.90 W;

③使第二个物理 CPU 中第一个计算单元满载时（启动第二个物理 CPU），系统功率增加约 35.58 W，为简化计算我们将 P_s 取②③的平均值 39.74 W;

④在物理 CPU 上启动 1 个全资源线程单元功率平均增加 P_c=11.13 W；

⑤而使一个共享线程单元满载功率平均增加 P_t=2.83 W。

因此可得出，基于表 1 的 OS 线程单元分配策略虽然性能提升很快，但功耗与成本增长也较快。

2.2 不同需求下的计算单元分配

表 4 计算单元分配要素

Table 4 The key elements of computing units allocation

			物理核心 →	Core 0	Core 1	Core 2	Core 3
物理处理器 ↓	Processor 0	线程单元 ↓	Thread 0	(1.0,35.61,1)	(1.0,9.68,1)	(1.0,9.68,1)	(1.0,9.68,1)
			Thread 1	(0.3,2.61,0)	(0.3,2.61,0)	(0.3,2.61,0)	(0.3,2.61,0)
	Processor 1		Thread 0	(1.0,14.77,1)	(1.0,9.68,1)	(1.0,9.68,1)	(1.0,9.68,1)
			Thread 1	(0.3,2.61,0)	(0.3,2.61,0)	(0.3,2.61,0)	(0.3,2.61,0)

为便于描述，选择结构相对简单的 Dell C6100 服务器为例进行分析。表 4 列出了该结点下的计算单元计算能力、功率及执行单元数，表 4 中用一个三元组（⊿计算能力，⊿功率，⊿物理计算核心数）描述每个计算单元的引入功效与成本。该三元组表示，如果要为计算分配该计算单元所能获得的计算能力、增加的功率及占用的物理计算单元数(共享线程单元的计算能力，按照综合经验值取 0.3，不需要增加物理计算单元)。这里假定每个物理处理器上总是从 Core0 开始分配全资源线程单元。

调度策略 1：加速比最大化需求

对于性能要求高或串行部分 S 比值较大的计算任务，由于受并行可扩展性的限制，应优先投入性能强的计算单元，从而可以使用较少数量的计算单元取得好的性能。

调度策略 2：“性能/功率/物理计算单元数”最大化需求

当用户对性能要求不高，而将节约计算中心的运维成本作为首要追求目标时，“功耗=运行总时间*运行功率”成为评价作业运行的重要指标。由于计算结点的基础功率 P_n 通常远大于物理计算单元的功率 P_c 及线程单元功率 P_t，通常在小规模计算中，同时投入的计算能力越多、越强，则计算的“性能/功率”比率越高，时间越短、总功耗也越少。虽然研究界与工业界多通过降低主频来降低功耗，但 Intel 的 Turbo Boost 就采用这种类似思想通过提升处理器主频来降低计算的功耗。

鉴于 SMT 的特点，本文在“性能/功率”比率的基础上引入 SPN=“加速比/功率/物理计算单元数”来描述单位计算单元的计算能力与功耗效率，并用以指导计算单元的分配：$\mathrm{SPN}=1000*\frac{\mathrm{SN}}{\mathrm{P}}/\mathrm{C}$

其中 SN 为加速比，P 为功率，C 为物理计算单元数，1000 为常系数。在分配计算单元时，以 SPN 最大化作为调度目标，从而提高单位功率、单位成本的计算效率。

2.3　投入计算单元数量上限

一个应用投入计算单元的数量一般由用户在提交作业时确定。在一定范围内投入的计算单元越多则计算时间越短。由于功耗 $W=P*t$，设初始计算方案功率为 P_0，任务执行时间为 t_0。而拟再投入一个计算单元，其功率为 ΔP，缩短计算时间 Δt，功耗变化为 ΔW。若仅从功耗角度考虑，是否在现有计算资源的基础上再投入新的计算资源，取决于：$(P_0+\Delta P)*(t_0-\Delta t)<P_0*t_0$ 是否成立。

即：$(t_0-\Delta t)/t_0<P_0/(P_0+\Delta P)$是否成立。设计算任务在单计算单元执行时间为 1，则多核并行的执行时间为：$\frac{1}{\mathrm{SN}}=\frac{(\mathrm{N-1})\ \mathrm{S+1}}{\mathrm{N}}$，可推出 $\mathrm{t}=\frac{1-\mathrm{S}}{\mathrm{N}\ (\mathrm{N-1})}$。

如果通过实验取得了一个加速比的值，那么从（公式-1）也可推算出一个计算中的 S：$S=(N-SN)/SN(N-1)$因此可依此对计算任务投入计算核心数的上限进行推算。

2.4　线程单元拓扑识别

在 OS 中可通过计算单元拓扑信息对线程单元间的关系进行识别，如 Linux 以 Core id，Physical id 等来提供处理器拓扑的相关信息：

1. 拥有相同 Physical id 的所有逻辑处理器共享同一个物理 Socket。每个 Physical id 代表一个唯一的物理封装。

2. 每个 Core id 代表一个唯一处理器核。

3. 如果有两个或两个以上的逻辑计算单元拥有相同 Physical id，且 Core id 不同，则说明这是一个多核处理器。

4. 如果有一个以上逻辑计算单元拥有相同的 Core id 和 Physical id，则说明系统支持 Hyper Threading，且这些逻辑计算单元在共享对应（Physical id，Core id）所指示物理计算核心的计算资源。

通过以上原则系统可以方便地探测系统处理器拓扑结构。如针对图 1 所示的 Hyper Threading 计算系统，Linux kernel 3.6.11 为其提供了如表 5 所示的处理器信息。表 5 第一行处理器编号即为 OS 对线程单元的编号，通常 OS 在空闲状态下会按此编号顺序依次向计算任务提供计算资源。

表 5 操作系统提供的处理器信息

Table 5 The processor information in operating system

处理器 Processor	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Physical id	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Core id	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

2.5 基于HT 拓扑感知与线程亲和性的SMT 调度机制

上述分配策略的具体实施可在应用层、运行时支持层或操作系统层上实现。

（1）在应用层实现仅影响该应用的计算单元分配策略，但此方案会增加应用程序员的编程负担；

（2）在运行时支持层实现则仅影响相关计算平台的使用[4,5]。

（3）在操作系统层实现，则计算单元分配策略会对全部应用产生影响。

由于线程的亲和性设定一般发生在计算的开始，因此以上三种方法在性能上差异不大，三种方法主要表现在对调度的影响范围不同。论文采用后两种方案来验证所提方法：在操作系统层，论文验证时通过动态修改 Linux 的进程亲和性系统调用[6]来修改操作系统的调度序列。在运行时支持层以斯坦福大学一研究小组基于共享存储 mapreduce 实现的 Phoenix 2.0[2]及 Phoenix++1.0[7]为案例进行验证。其中 Phoenix 2.0 在 src/processor.c 中设置线程亲和性，Phoenix ++1.0 在 include/processor.h 中设置亲和性，这二个模块均与用户逻辑无关。

3 测试与验证

分别在 Dell C6100（双路 Xeon L5520，24 G 内存，Linux kernel 3.6.11）及 HP DK388P（双路 XeonE5-2667，80 G 内存，Linux 2.6.32）上运行共享存储模式 Phoenix 的 word counter 案例并记录论文所提方案的执行情况。其中 Dell 结点运行 Phoenix 2.0 word counter 1600M 案例（原始程序未做优化），HP 结点由于性能显著优于 Dell C6100，因此运行 Phoenix ++1.0 word counter 51200 M 案例。在 Dell C6100 服务器上的实验结果如表 6 所示。

实验数据显示两类调度策略依据调度的目标不同，在性能和效率方面表现出较大的差异，适合不同需求的用户根据需要进行选择，克服了传统 OS 单一调度策略的不足。尤其在“性能/功率/物理计算单元数”这项指标上，“调度策略 2”充分利用了 Hyper Threading 的架构特点，使得每物理计算单元的效率得以充分发挥，在功耗方面取得不错的表现。

本文实验中，在功率与“调度策略 1”相当的情况下，基于“调度策略 2”中的每个物理核心（表 6 中偶数个线程单元）相对于“调度策略 1”中一个物理核心的计算性能最高提高了 33%左右（如表 6 中“调度策略 2”的 6 个共享线程单元与“调度策略 1”的 3 个全资源线程单元），平均提高 21%，当全部计算单元投入后，计算性能/功率/效率相当。

表 6 实验结果

Table 6 The result from experiment

线程数 Thread		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
执行时间	策略1	77.06	40.46	29.39	22.86	18.29	15.65	14.36	13.39	13.57	13.41	12.77	13.54	13.61	13.83	13.46	13.89
	策略2	76.68	34.60	32.35	27.58	21.55	19.67	16.78	15.51	14.62	14.20	13.55	13.57	13.27	13.28	13.60	13.37
功率	策略1	111.71	122.67	130.39	141.38	156.15	166.83	174.25	184.53	187.66	190.09	192.37	194.42	197.89	199.77	202.94	205.45
	策略2	110.34	113.32	127.08	130.13	142.53	146.03	158.15	161.36	168.42	171.18	176.78	179.84	190.96	193.60	204.41	207.27
加速比	策略1	1.00	1.90	2.62	3.37	4.21	4.92	5.37	5.76	5.68	5.75	6.03	5.69	5.66	5.57	5.73	5.55
	策略2	1.00	1.40	2.37	2.78	3.56	3.90	4.57	4.85	5.24	5.40	5.66	5.65	5.78	5.77	5.64	5.74
加速比/功率/单元数	策略1	8.95	7.76	6.70	5.96	5.40	4.92	4.40	3.90	3.78	3.78	3.92	3.66	3.58	3.49	3.53	3.38
	策略2	9.06	12.38	9.33	10.68	8.32	8.90	7.22	7.51	6.23	6.31	5.34	5.24	4.32	4.26	3.45	3.46

在功耗方面，当“调度策略 2”使用 4 个共享线程单元功率时为 130.13 W，与“调度策略 1”的 3 个全资源线程单元功率相当，但执行速度要快于“调度策略 1”；“调度策略 2”使用 10 个线程单元执行时间为 171.18 s，与“调度策略 1”的 7 个线程单元时间相当，但功率却比“调度策略 1”降低了 3 W。同样“调度策略 2”使用 12 个线程单元执行时间与“调度策略 1”的 9 个线程单元时间相当，但功率却

比“调度策略 1”降低了 7.82 W。表 7 以 8 个物理核心为单位对表 6 中的数据进行了对比。图 2、3、4 分别为 HP DK388P 服务器上时间、“性能/功率/物理计算单元数”、功率三项指标，在“调度策略 1”与“调度策略 2”两种策略下的比较图示。限于篇幅数据表格不再列出。

表 7 实验结果分析

Table 7 The analysis on experimental results

处理核心数 Cores		1	2	3	4	5	6	7	8
执行时间	策略1	77.06	40.46	29.39	22.86	18.29	15.65	14.36	13.39
	策略2	54.66	27.58	19.67	15.81	14.20	13.57	13.28	13.37
策略2/策略1		0.71	0.68	0.67	0.69	0.78	0.87	0.92	1
功率	策略1	111.71	122.67	130.39	141.38	156.15	166.83	174.25	184.53
	策略2	113.32	130.13	146.03	161.36	171.18	179.84	193.60	207.27
（策略2/策略1）-1		0.01	0.06	0.12	0.14	0.10	0.08	0.11	0.12
加速比	策略1	1.00	1.90	2.62	3.37	4.21	4.92	5.37	5.76
	策略2	1.40	2.78	3.90	4.85	5.40	5.65	5.77	5.74
加速比/功率/单元数	策略1	8.95	7.76	6.70	5.96	5.40	4.92	4.40	3.90
	策略2	12.38	10.68	8.90	7.51	6.31	5.24	4.26	3.46

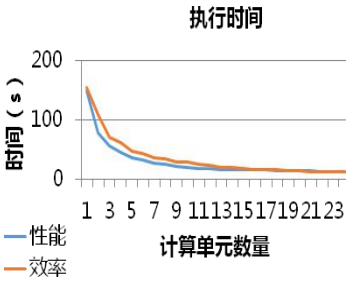

图 2 HPDK388P 服务器计算执行时间对比

Fig.2 Comparison of execution time on server HPDK388P

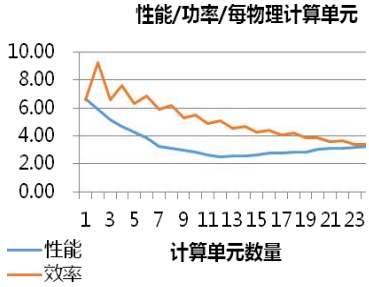

图 3 HPDK388P 服务器计算性能/功率/物理计算单元对比

Fig.3 Comparison of performance, power and physics computing elements on server

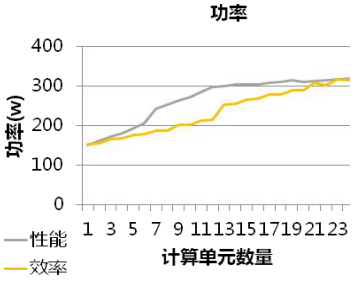

图 4 HPDK388P 服务器计算功率对比

Fig.4 Comparison of powers on server HPDK388P

图 2~4 显示 HP DK388P 服务器在性能、功率及"加速比/功率/物理计算单元数"具有与 DELL C6100 基本相似的特性。

综上所述，论文提出的面向“性能/功率/物理计算单元数”的调度策略充分挖掘了 Hyper Threading 架构的特点，有效提高了计算系统的计算效率。

4　结束语

本文分析了现有计算操作系统对 Hyper Threading 技术支持的不足之处，提出了利用线程亲和性调整线程单元的分配顺序的调度策略，有效挖掘 SMT 技术的 ILP 与 TLP；所提方法可以在应用层、在运行时支持与操作系统层灵活部署，用以满足不同用户计算任务的需求，从而有效地提高了系统计算资源利用率，并使得并行计算任务可以用更低的功耗获得更高的性能。

参考文献

[1] 林 闯,田 源,姚 敏.绿色网络和绿色评价:节能机制、模型和评价[J].计算机学报,2011,34(4):593-612

[2] Richard M. Yoo, Romano A, Kozyrakis C. Phoenix rebirth: Scalable Mapreduce on a large-scale shared-memory system[C].USA:IEEE International Symposium on Workload Characterization, 2009:198-207

[3] 冯国富,王 明,李 亮,等.共享存储 MapReduce 云计算性能测试方法[J].计算机工程,2012,38(6):50-52

[4] 冯国富,董小社,胡 冰,等.一种支持多种访存技术的 CBEA 片上多核 MPI 并行编程模型[J].计算机学报,2008(11):1965-1974

[5] 董小社,冯国富,王旭昊,等.基于 Cell 多核处理器的层次化运行时支持技术[J].计算机研究与发展,2010(4):561-570

[6] 冯国富,魏恒义,储 鹰,等.支持多种 Linux 版本的动态内核性能测试技术[J].西安交通大学学报,2008(6):674-678

[7] Talbot J, Yoo RM, Kozyrakis C. Phoenix++: Modular MapReduce for Shared-Memory Systems[C]. San Jose, CA:the Second International Workshop on Mapreduce and its Applications, 2011:9-16