面向众核密码处理器的高效负载均衡技术

戴紫彬[①] 尹安琪*[①] 曲彤洲[①] 南龙梅[①②]

[①](解放军信息工程大学 郑州 450001)

[②](复旦大学专用集成电路与系统国家重点实验室 上海 201203)

摘 要：工作负载分配不均是制约众核密码平台资源利用率提高的重要因素，动态负载分配可提高平台资源利用率，但具有一定开销；所以更高的负载均衡频率并不一定带来更高的负载均衡增益。因此，该文建立了关于负载均衡增益率与负载均衡频率的数学模型。基于模型，提出一种面向众核密码平台的无冲突负载均衡策略和一种基于硬件作业队列的“可扩展-可移植”负载均衡引擎——“簇间微网络-簇内环阵列”。实验证明：在性能、延时功耗积、资源利用率和负载均衡度方面，该文设计的负载均衡引擎与基于“作业窃取”的软件技术相比平均优化约4.06倍、7.17倍、23.01%和2.15倍；与基于“作业窃取”的硬件技术相比约优化1.75倍、2.45倍、10.2%、和1.41倍；与理想硬件技术相比，密码算法吞吐率平均只降低了约5.67%(最低3%)。实验结果表明该文技术具有良好的可扩展性和可移植性。

关键词：众核密码处理器；负载均衡策略；负载均衡引擎；无冲突

中图分类号：TP338.6 **文献标识码**：A **文章编号**：1009-5896(2019)02-0369-08

DOI: 10.11999/JEIT180623

Efficient Workload Balance Technology on Many-core Crypto Processor

DAI Zibin[①] YIN Anqi[①] QU Tongzhou[①] NAN Longmei[①②]

[①](*The PLA Information Engineering University*, *Zhengzhou* 450001, *China*)

[②](*State Key Laboratory of ASIC and System*, *Fudan University*, *Shanghai* 201203, *China*)

Abstract: Imbalanced workload distribution results in low resource utilization of many-core crypto-platform. Dynamic workload allocation can improve the resource utilization with some overhead. Therefore, a higher frequency of workload balancing is not equivalent to higher gains. This paper establishes a mathematical model for gain rate and frequency of workload balancing. Based on this model, a collision-free workload balancing policy is proposed for many-core crypto systems, and a hierarchical "expandable-portable" engine is put forward, which consists of "Inter-cluster micro-network and intra-cluster ring-array" adopting hardware job queue technology. Experiment results show that the proposed workload-balancing engine is 4.06, 7.17, 23.01% and 2.15 times higher than the software technology based on “job stealing” in terms of performance, delay power consumption, resource utilization and workload balance; 1.75, 2.45, 10.2%, and 1.41 times better compared with the hardware technology based on "job stealing". By contrast with the ideal hardware technology, the average throughput of encryption algorithms is only decreased by 5.67% (the lowest 3%). The experiment also proves the scalability and portability of the proposed technique.

Key words: Many-core crypto processor; Workload balance strategy; Workload balance engine; Collision-free

1 引言

密码应用是典型的计算密集型应用，而众核平台并行资源丰富，理论上可以大幅提升其实现性能；但负载不均衡问题导致平台资源利用率低，难以实现理想的性能提升。线程调度[1,2]和作业队列技术[3–6]是通用领域实现负载均衡的有效方式。对于具有极强数据相关性的密码计算来说[7,8]，过度开发其线程级并行性会带来显著的通信开销；且随着平台计算粒度的降低，线程本身的执行时间会不断缩短，而线程创建及页面切换的开销占比会不断增大[9,10]。这将折损线程并行带来的性能增益，但作业队列技术不存在线程创建与页面切换问题。这里明确线程是指一个程序的指令及其执行状态，而作业是指一个线程执行的数据集合[3]。密码应用的数据相关性使其分配方案依赖于输入数据(与通用

收稿日期：2018-06-26；改回日期：2018-11-27；网络出版：2018-12-03

*通信作者： 尹安琪 yinaq0222@foxmail.com

领域中任务级并行应用[11]相似)，而实际应用场景中输入数据大多是随机的，无法在编译阶段确定。软件作业队列技术操作延时较大，较难实现高效的动态负载分配[12]。为此文献[5]设计了一种放置于通信网络中的硬件队列控制器，以加速动态负载分配。

基于作业队列的负载均衡技术具有冲突问题——不同线程访问相同数据结构时就会产生冲突，这实际上构成了其性能瓶颈[13]。分布式队列不同线程独享各自的队列，与集中式相比极大降低了冲突频率[4]。文献[4]就此提出了一种分布式硬件队列管理技术Carbon，通过“作业窃取”的方式实现负载均衡，实现了较为理想的性能提升，但只降低了冲突频率，未能根除冲突问题。

作业队列技术的另一典型问题是可扩展性问题，即当核心数目增多时，性能增益锐减，对于硬件作业队列来说这还会造成硬件开销激增问题[14]。为此文献[3]提出了一种基于可扩展数据传输网络的作业队列技术，提高了作业队列技术的可扩展性。也有文献[6]考虑到核心数目增多时数据传输网络的可靠性问题，设计了一种容错网络，进一步提高了作业队列技术的可扩展性。在传统的基于复杂超标量体系结构的平台上实现密码应用时，往往只关注实现性能；但不同的众核平台在架构及互连方式等方面具有较大差异，上述技术在解决负载均衡问题时均未考虑技术的可移植性[6]。

由于动态负载分配具有一定开销，更高的负载均衡频率并不一定带来更高的负载均衡增益，所以本文建立了关于负载均衡增益率和负载均衡频率的数学模型。基于模型提出一种无冲突负载均衡策略，并设计一种基于环形硬件队列的“可扩展-可移植”负载均衡引擎，可解决作业队列的冲突、可扩展性以及可移植性问题，提高了众核密码平台的负载均衡度和密码算法实现性能。

2 数学模型

2.1 众核密码处理器建模

为支持负载均衡技术的可扩展性与可移植性研究，本文建立了如图1所示的众核密码处理器架构模型。

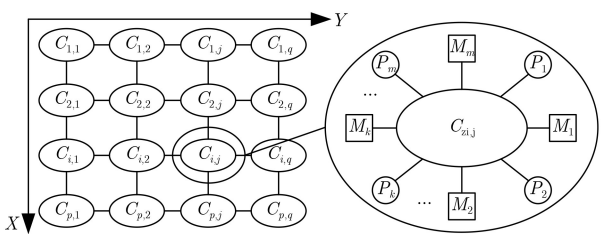


图 1 众核密码处理器架构模型

$C_{(i,j)}$表示一个具体的簇，其中(i,j)表示簇坐标；$P_{(i,j,k)}$，$M_{(i,j,k)}$分别表示簇$C_{(i,j)}$内的一个具体的核心和共享存储，k为簇内坐标。通信方式为簇内共享存储，簇间片上网络。众核密码处理器MCP可以抽象成

$$\left.\begin{aligned}\mathrm{MCP}\,(p,q,m)&=\sum_{i=1}^{p}\sum_{j=1}^{q}C_{(i,j)}\\ C_{(i,j)}&=\sum_{k=1}^{m}\left(P_{(i,j,k)}+M_{(i,j,k)}\right)\end{aligned}\right\}\tag{1}$$

其中，参数p, q, m可以表征密码平台的属性：(1)可扩展性：p, q, m均为变量，所以簇数与核心数目是可扩展的；(2)兼容性：可兼容分簇式/非分簇式架构$(m=1/m\neq 1)$以及不同的簇间互连方式；(3)量化性：令并行因子$\alpha=p\times q\times m\geq 1$, α可以定量衡量平台的并行资源大小。

2.2 关于负载均衡增益率与负载均衡频率的数学模型

2.2.1 模型参数定义

设某作业job_x的执行时间为T_{jb_x}，约是其所在线程的线程长度L_{td_x}，线程循环次数N_{cl_x}及作业长度L_{dt_x}的乘积，即

$$T_{\mathrm{jb}_x}=L_{\mathrm{td}_x}\times N_{\mathrm{cl}_x}\times L_{\mathrm{dt}_x}\tag{2}$$

设一个核心的总执行时间为$T_{\mathrm{tl}}^{P_{(i,j,k)}}$，包括作业执行时间$T_{\mathrm{ex}}^{P_{(i,j,k)}}$，等待时间$T_{\mathrm{wt}}^{P_{(i,j,k)}}$与空闲时间$T_{\mathrm{id}}^{P_{(i,j,k)}}$，其中$T_{\mathrm{ex}}^{P_{(i,j,k)}}$为分配至核$P_{(i,j,k)}$的作业集合$\mathrm{job}^{P_{(i,j,k)}}$的执行时间。即

$$\left.\begin{aligned}T_{\mathrm{tl}}^{P_{(i,j,k)}}&=T_{\mathrm{ex}}^{P_{(i,j,k)}}+T_{\mathrm{wt}}^{P_{(i,j,k)}}+T_{\mathrm{id}}^{P_{(i,j,k)}}\\ T_{\mathrm{ex}}^{P_{(i,j,k)}}&=\sum_{\mathrm{job}_x\in\mathrm{job}^{P_{(i,j,k)}}}T_{\mathrm{job}_x}\end{aligned}\right\}\tag{3}$$

设参与负载均衡操作的核心构成负载均衡集合S。S的总执行时间T_{tl}^{S}是S中最后执行完毕核心P_{lt}的总执行时间(P_{lt}的空闲时间为0)，也就是S中所有核心执行时间的最大值，即

$$\left.\begin{aligned}T_{\mathrm{tl}}^{S}&=T_{\mathrm{tl}}^{P_{\mathrm{lt}}}=\max\left\{T_{\mathrm{tl}}^{P_{(i,j,k)}}\right\},P_{(i,j,k)}\in S\\ T_{\mathrm{tl}}^{P_{\mathrm{lt}}}&=T_{\mathrm{ex}}^{P_{\mathrm{lt}}}+T_{\mathrm{wt}}^{P_{\mathrm{lt}}}\end{aligned}\right\}\tag{4}$$

设负载均衡操作的时间轴为$t(t_0\leq t\leq t_{\mathrm{ed}})$。定义$(t_0,t_{\mathrm{ed}})$时间内，集合$S$的负载均衡增益率$\eta$为$t_0$时刻与$t_{\mathrm{ed}}$时刻集合$S$总执行时间的比值。根据式(4)，$\eta$表达式为

$$\eta=\frac{T_{\mathrm{tl\_}t_0}^{S}}{T_{\mathrm{tl\_}t_{\mathrm{end}}}^{S}}=\frac{\max\left\{T_{\mathrm{tl\_}t_0}^{P_{(i,j,k)}}\right\}}{\max\left\{T_{\mathrm{tl\_}t_{\mathrm{ed}}}^{P_{(i,j,k)}}\right\}},P_{(i,j,k)}\in S\tag{5}$$

对于单次均衡操作，S只包含收发核心P_{re}, P_{sd}

两个元素，即$S=\{P_{\mathrm{re}},P_{\mathrm{sd}}\}\,(T_{\mathrm{ex}}^{P_{\mathrm{sd}}}>T_{\mathrm{ex}}^{P_{\mathrm{re}}})$。设均衡周期为$c_b$，某次均衡操作起于$t_y$，止于$t_{y+c_b}$。根据式(5)，$t_y$时刻均衡操的负载均衡增益率$\eta_{t_y}$为

$$\eta_{t_y}=\frac{T_{\mathrm{tl}\_t_y}^{S}}{T_{\mathrm{tl}\_t_{y+c_b}}^{S}}=\frac{T_{\mathrm{tl}\_t_{\mathrm{y}}}^{P_{\mathrm{sd}}}}{\max\left\{T_{\mathrm{tl}\_t_{y+c_b}}^{P_{\mathrm{sd}}},T_{\mathrm{tl}\_t_{y+c_b}}^{P_{\mathrm{re}}}\right\}}\tag{6}$$

2.2.2 模型建立

由于负载均衡操作会带来一定的开销，更高的负载均衡频率并不一定带来更高的负载均衡增益，本文模型旨在求解如何提高负载均衡增益。

负载均衡开销T_b包括：(1)负载计算开销T_{op}，用于判断是否需要进行负载均衡操作和具体的作业收发核心P_{re}, P_{sd}，(2)线程及作业传输开销T_{tr}。设平台的数据传输吞吐率为Thput，则

$$T_b=T_{\mathrm{cp}}+T_{\mathrm{tr}}=T_{\mathrm{cp}}+(L_{\mathrm{dt}}+L_{\mathrm{td}})/\mathrm{Thput}\tag{7}$$

均衡操作带来性能提高的充要条件是执行所有均衡操作后负载均衡增益率大于1，即

$$\eta=\prod_{t=t_0}^{t_{\mathrm{ed}}}\eta_t>1\tag{8}$$

进一步，其充分条件是每次负载均衡操作的负载均衡增益率均大于1，即

$$\bigcap_{t=t_0}^{t_{\mathrm{ed}}}\eta_t>1\tag{9}$$

以下求解式(9)，设一次负载均衡操作起于t_y，止于t_{y+c_b}；待分配作业为$\mathrm{job}_b^{t_y}$。根据式(6)，t_y时刻负载均衡操的负载均衡增益率η_{t_y}为

$$\left.\begin{aligned}\eta_{t_y}&=\frac{T_{\mathrm{tl}}^{P_{\mathrm{sd}}\_t_y}}{\max\left\{T_{\mathrm{tl}}^{P_{\mathrm{sd}}\_t_{y+c_b}},T_{\mathrm{tl}}^{P_{\mathrm{re}}\_t_{y+c_b}}\right\}}\\T_{\mathrm{tl}}^{P_{\mathrm{sd}}\_t_{y+c_b}}&=T_{\mathrm{tl}}^{P_{\mathrm{sd}}\_t_y}-T_{\mathrm{job}_b^{t_y}}\\T_{\mathrm{tl}}^{P_{\mathrm{re}}\_t_{y+c_b}}&=T_{\mathrm{tl}}^{P_{\mathrm{re}}\_t_y}+T_{\mathrm{job}_b^{t_y}}+T_b\end{aligned}\right\}\tag{10}$$

又$T_{\mathrm{tl}}^{P_{\mathrm{sd}}_t_y}>T_{\mathrm{tl}}^{P_{\mathrm{sd}}_t_y}-T_{\mathrm{job}_b^{t_y}}$，由式(2)，式(7)，式(10)可得到一次负载均衡增益率大于1的充分条件是

$$T_{\mathrm{tl}}^{P_{\mathrm{sd}}\_t_y}-T_{\mathrm{tl}}^{P_{\mathrm{re}}\_t_y}-L_{\mathrm{td}_b^{t_y}}\cdot N_{\mathrm{cl}_b^{t_y}}\cdot L_{\mathrm{dt}_b^{t_y}}>T_{\mathrm{cp}}^{t_y}+T_{\mathrm{tr}}^{t_y}\tag{11}$$

具体到2.1节的众核平台模型，若负载均衡操作发生在簇内，则不需要进行线程与数据的传输，即T_{tr}为0。由式(11)得，簇内$\eta>1$的充分条件是

$$\bigcap_{t=t_0}^{t_{\mathrm{ed}}}T_{\mathrm{tl}}^{P_{\mathrm{sd}}\_t}-T_{\mathrm{tl}}^{P_{\mathrm{re}}\_t}-L_{\mathrm{td}_b^t}\cdot N_{\mathrm{cl}_b^t}\cdot L_{\mathrm{dt}_b^t}>T_{\mathrm{cp}}^{t}\tag{12}$$

若负载均衡操作发生在簇间，因平台数据传输吞吐率Thput受到存储体吞吐率$\mathrm{Thput}_{\mathrm{SM}}$和通信网络吞吐率的$\mathrm{Thput}_{\mathrm{Nt}}$的约束，所以由式(9)，式(11)，簇间$\eta>1$的充分条件是

$$\bigcap_{t=t_0}^{t_{\mathrm{ed}}}T_{\mathrm{tl}}^{P_{\mathrm{sd}}\_t}-T_{\mathrm{tl}}^{P_{\mathrm{re}}\_t}-L_{\mathrm{td}_b^t}\cdot N_{\mathrm{cl}_b^t}\cdot L_{\mathrm{dt}_b^t}>T_{\mathrm{cp}}^{t}+\frac{L_{\mathrm{dt}}^t+L_{\mathrm{td}}^t}{\mathrm{Thput}_{\mathrm{SM}}},\mathrm{Thput}_{\mathrm{SM}}\le\mathrm{Thput}_{\mathrm{Nt}}\tag{13a}$$

$$\bigcap_{t=t_0}^{t_{\mathrm{ed}}}T_{\mathrm{tl}}^{P_{\mathrm{sd}}\_t}-T_{\mathrm{tl}}^{P_{\mathrm{re}}\_t}-L_{\mathrm{td}_b^t}\cdot N_{\mathrm{cl}_b^t}\cdot L_{\mathrm{dt}_b^t}>T_{\mathrm{cp}}^{t}+\frac{L_{\mathrm{dt}}^t+L_{\mathrm{td}}^t}{\mathrm{Thput}_{\mathrm{Nt}}},\mathrm{Thput}_{\mathrm{SM}}>\mathrm{Thput}_{\mathrm{Nt}}\tag{13b}$$

密码计算与均衡操作的执行有串行、流水与并行3种方式。设密码计算与负载均衡操作的执行周期分别为c_e与c_b，单位负载均衡周期内启动的负载均衡操作的次数为N_b。3种方式下的密码计算频率F_e和负载均衡频率F_b分别如式14(a)，式14(b)和式14(c)所示

$$F_e=1/(c_e+c_b),F_b=N_b/(c_e+c_b)\tag{14a}$$

$$F_e=\min\{1/c_e,1/c_b\},\quad F_b=N_b\times\min\{1/c_e,1/c_b\}\tag{14b}$$

$$F_e=1/c_e,\qquad F_b=N_b/c_b\tag{14c}$$

2.2.3 模型分析

从模型可知，负载均衡操作虽然可以减小核心空闲时间$T_{\mathrm{id}}^{P_{(i,j,k)}}$，但会增加核心等待时间$T_{\mathrm{wt}}^{P_{(i,j,k)}}$；只有每次均衡操作均可带来性能增益，提高负载均衡频率才一定带来负载均衡增益率的提高。所以若每次负载均衡增益率η_t都大于1(式(9))，负载均衡增益便与负载均衡频率呈正相关。

负载均衡操作若发生在簇内，η_t的计算只需考虑负载均衡的计算开销，那么满足式(12)即可实现$\forall\eta_t>1$；若发生在簇间则还要考虑任务和数据的传输开销。当存储体吞吐率小于通信网络吞吐率时，要满足式(13a)的约束；反之，要满足式(13b)的约束。进一步，由式(14a)，式(14b)，式(14c)，提高负载均衡频率可以从减小负载均衡操作延时与提高单位均衡周期内启动的均衡操作的次数N_b两个方面考虑。

此外根据式(5)，要优先执行具有最大延时核心的作业再分配，才能减小密码任务的总执行时间，从而带来性能提升。最后，所提出的负载均衡技术应使负载均衡操作与密码计算并行执行(式

(14a)，式(14b)，式(14c))。这是因为串行作业下，密码计算和负载均衡操作相互恶化对方的性能，因而具有最低的F_b；流水作业下，若$c_e > c_b$，密码计算延时会制约F_b的提高；反之，负载均衡操作会恶化密码计算的性能；并行作业下，二者互不影响。

3 负载均衡技术

3.1 无冲突负载均衡算法

为解决作业队列技术存在的冲突问题，本文提出了一种“作业给予”式的无冲突负载均衡策略。表1以簇间为例，给出了无冲突负载均衡算法的伪代码。算法步骤如下：(1)只要检测到新作业执行请求，就更新簇负载累加值(2～5行)；(2)判断存储体吞吐率是否大于通信网络吞吐率(10行)；(3)S_c中任意一个簇与其他簇进行均衡增益决策，若(2)的判断结果为真则根据式13(a)决定是否向对方发送作业传输请求(11～14行)；(4)若(2)的判断结果为假，则根据式13(b)进行决策(15～20行)；(5)若一个簇接收到多个作业传输请求，采用轮询的方式接受作业，否则不进行任何操作(22～26行)。簇内与此相似，但均衡增益决策由式(12)决定且不受存储体与网络吞吐率约束。

表 1 无冲突负载均衡算法

Require: $L_{\rm td}$, $N_{\rm cl}$, $L_{\rm dt}$, $T_{\rm cp}$, $\rm Thput_{SM}$, $\rm Thput_{Nt}$, $S_{\rm c}$//簇间均衡操作集合

1 Assign Balance_c $\leftarrow$ True, $N_2 \leftarrow \rm Num[S_c]$, $n_2 \leftarrow 0$, $t_2 \leftarrow T_{\rm op}$ $t_3 \leftarrow (L_{\rm td}+L_{\rm dt})/\rm Thput_{SM}$, $t_4 \leftarrow (L_{\rm td}+L_{\rm dt})/\rm Thput_{\mathit{Nt}}$

2 **while** new job$[k][i]$ == True **do**//更新负载情况

3 $t_1 \leftarrow L_{\rm tdki} \cdot N_{\rm clki} \cdot L_{\rm dtki}$; Addc$[k]$ $\leftarrow$ Addc$[k]$ $+ t_1$;

4 **end while**

5 **while** Balance_c == True **do**//簇间负载均衡

6 **for** $k \leftarrow 0$ to N_2 **do**

7 **for** $w \leftarrow 0$ to N_2 **do**

8 **if** $w \neq k$ **then**

9 **if** $\rm Thput_{SM} \geq Thput_{\mathit{Nt}}$ **then**

10 **if** Addp$[k]$ − Addp$[w]$−t_1>t_2+t_4 **then//式(13)**

11 Balancep_Request$[w][k] \leftarrow$ True;

12 Balancec_Request$[w]$ = Balancec_Request$[w]$ + 1;

13 **end if**

14 **else then**

15 **if** Addp$[k]$ − Addp$[w]$−t_1 > t_2 + t_3 **then//式(13)**

16 Balancep_Request$[w][k] \leftarrow$ True;

17 Balancec_Request$[w]$ = Balancec_Request$[w]$ + 1;

18 **end if**

19 **end else**

20 **end for**

21 **if** Balancec_Request$[w]$ > 1 **then**

22 choose p[n$_2$]; $n_2 \leftarrow (n_2+1) \rm mol N_2$;

23 **end if**

24 **end for**

25 **end while**

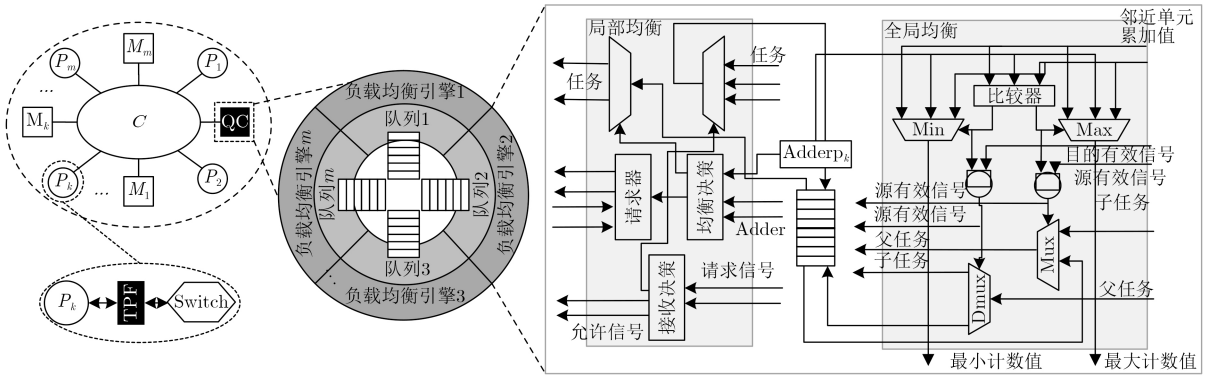
3.2 负载均衡引擎

基于密码任务的动态负载分配需求，本文采用硬件实现3.1节中的算法，以减小均衡操作的延时开销。硬件设计包括3个部分：硬件作业队列架构、簇内负载均衡引擎阵列与簇间负载均衡微网络。

3.2.1 硬件作业队列架构

根据图1的平台模型与第1节中降低冲突频率的应用需求，本文设计的作业队列架构整体上是分簇式的，簇内是同构的且是分布式的，其中簇内架构如图2(a)所示。

根据式(12)，为降低作业传输开销$T_{\rm tr}$，硬件实现的作业队列在物理上相邻且成环状分布，其与负载均衡引擎阵列均置于簇队列管理中心(QC)。为了隐藏访问硬件作业队列的延时开销，从而降低负载均衡计算开销$T_{\rm cp}$，在每个核心内添加了任务预取单元TPF。根据式(14a)，式(14b)，式(14c)，将



(a) 簇内作业队列架构 (b) 簇内负载均衡引擎阵列架构 (c) 单个负载均衡引擎块电路

图 2 簇内作业队列架构

作业队列设计为双时钟队列，以隔离密码计算与负载均衡计算。

3.2.2 簇内负载均衡引擎阵列

基于3.1节中的无冲突负载均衡算法，设计了图2(b)所示的负载均衡引擎阵列架构以及图2(c)所示的引擎单元b_k。b_k包括全局均衡模块、局部均衡模块、作业队列及簇内负载累加器。b_k电路不会随m变，具有结构上的可扩展性。任选一个引擎单元作为根单元，用b_r表示。以$r=\lfloor m/2 \rfloor$为例，若$k<r$，则b_{k-1}是b_k的子单元，b_{k+1}是b_k的父单元；若$k>r$，则反之。b_k及其子单元构成的集合为b_k的子集合。

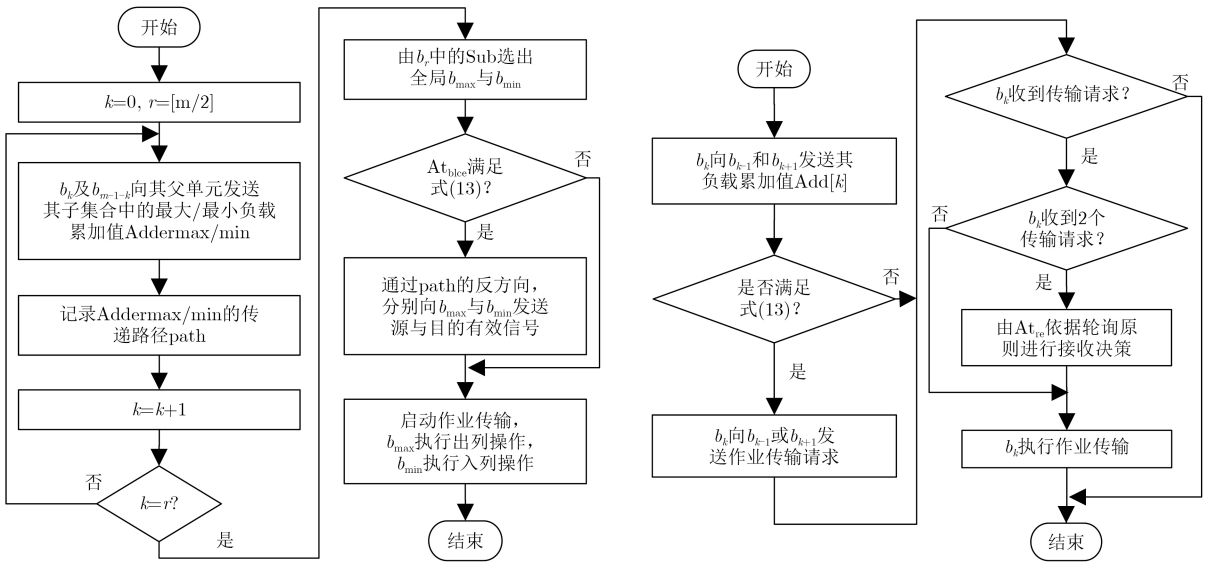
(1)全局均衡模块 根据式(5)，只有减小具有最大延时核心的执行延时，才能减小密码任务的总执行时间。全局均衡模块的设计目的即是找到全局最大和最小负载单元(b_{mx}和b_{mn})，因此设计了选择器Max和Min。此外b_{r}还包含减法电路Sub与均衡决策模块$\mathrm{At}_{\mathrm{bc}}$——根据式(12)设计的硬件电路。根据3.2节提出的负载均衡策略，全局均衡的执行流程如图3(a)所示。

(2)局部均衡模块 根据式14(c)，可以提高单位均衡数次N_b来提高负载均衡频率F_b。因此设计了局部均衡模块。此时$S=\{b_{k-1},b_k,b_{k+1}\}\ (0\le k<m)$，每个$b_k$都需要进行负载均衡决策，因此将$\mathrm{At}_{\mathrm{bc}}$添加到各个$b_k$中。局部均衡模块还包含依据轮询方式设计的接收决策模块$\mathrm{At}_{\mathrm{re}}$。具体的局部均衡执行流程如图3(b)所示。

3.2.3 簇间负载均衡微网络

根据式(14a)，式(14b)，式(14c)，设计了凌驾于现有通信网络之上的簇间负载均衡引擎微网络，如图4所示。

设计微网络时将众核密码处理器视为黑盒，所以微网络可移植到不同的众核平台上。微网络是一种柱形网络，其节点与簇内引擎单元b_k不同之处有



(a) 全局负载均衡执行流程　　(b) 局部负载均衡执行流程

图 3 全局及局部均衡操作执行流程

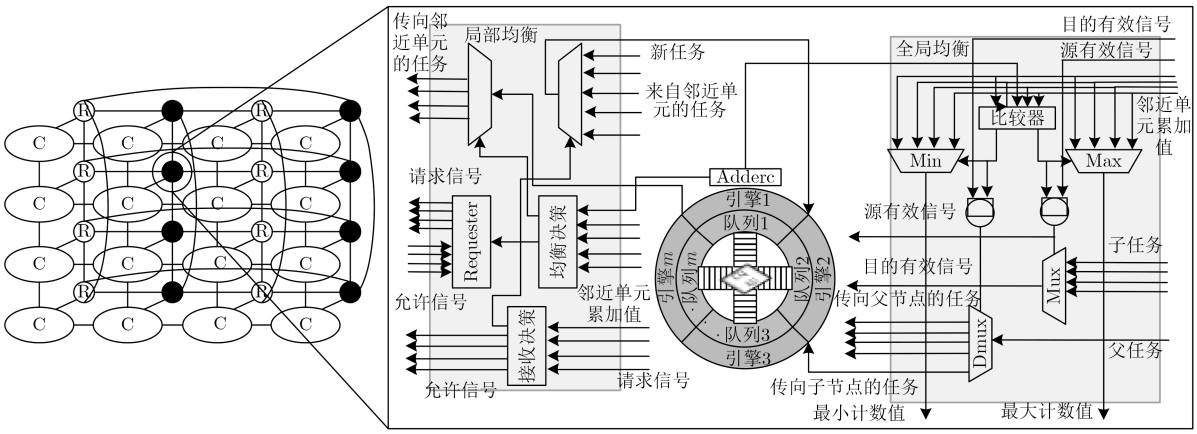


图 4 簇间负载均衡微网络

以下几点：(1)增加了1个簇间负载累加器；(2)增加了1个簇内队列仲裁模块CQAU ，以仲裁来自簇外的作业具体输入到簇内的哪个队列以及具体将簇内哪个队列的作业输出到簇外；出/入列仲裁分别采用负载最大/最小原则；(3)仲裁模块的输入端口数目由2变为4。

4 设计实现及评估

4.1 仿真平台搭建

在VS2010上，采用SystemC语言搭建了图1中的众核密码平台数据传输系统框架，可以实现作业的输入，输出，传输，滞留以及清空，表2给出了相应的仿真参数。为评估所提出负载均衡引擎的面积与功耗，采用Verilog语言描述该技术，并在55 nm工艺下实现ASIC设计流程。使用Synopsys Design Compiler进行综合，分析面积开销，并使用Synopsys Prime Time执行功耗分析。

表 2 仿真参数

参数	值
核心数目	4~64
本地缓存大小	8 kB
共享缓存大小	16 MB
簇内互连方式	二向环
簇间互连方式	2D-mesh
共享访问延时/本地访问延时	3

参与比较的负载均衡技术有4种：(1)基于任务窃取的软件负载均衡技术[15]；(2)Carbon[4]，一种典型的基于任务窃取的硬件负载均衡技术，已被证明具有高效性和良好的可扩展性，实验中对其进行适当改进，使其适于解决众核密码处理器上的负载均衡问题；(3)本文提出的负载均衡引擎；(4)理想的负载均衡引擎。其中(4)是在(3)基础上忽略一些硬件约束实现的：忽略硬件队列存储空间大小的约束；将访问硬件队列的延时设置为1个时钟周期；假设配置页面被缓存在本地存储中，并且可以在3个周期内完成配置。为了实现公平对比，使用SystemC语言描述所有负载均衡技术，并置于相同的仿真环境进行仿真。

本文选取的测试基准包括：(1)单密码算法：分组，杂凑，序列以及非对称密码算法；(2)多密码算法：对单密码算法进行随机组合。所使用单密码算法的测试基准与数据集合如表3所示。

4.2 评估

4.2.1 关键路径及设计开销评估

实验得到负载均衡引擎的最大时钟频率是74.4 MHz，单个簇对应的负载均衡引擎的面积和功耗分别约为0.0227 mm^2和0.246 mW。本实验室现有一款多核密码处理器MUP，假设众核密码处理器MCP1个簇的计算能力与原MUP1个核心相当；并假设MCP核心面积S_{MC}与MUP核心面积S_{MU}的之间的关系是S_{MC}=1.2×(S_{MU})/m，其中1.2是一个估计的面积增大因子，功耗作类似估计。那么，当$m=4$时，MCP1个簇的面积约为5.4 mm^2，功耗约为0.8 W。所以本文提出的负载均衡引擎的面积与功耗占比分别约为0.41%与0.031%；即使与通用平台(Cortex-A5[3])相比，面积与功耗开销占比也只有1.84%和0.01%左右。

4.2.2 性能评估

通过密码算法吞吐率来衡量其性能。具体地，本文探究α及测试基准变化时，各种技术的相对性能。图5以$\alpha=64$时理想技术下分组密码算法吞吐率为参考，对各吞吐率作了标准化处理。

软件与硬件对比：由图5(a)，硬件技术整体上要优于软件技术，且其性能具有更优的可扩展性。当$\alpha=64$时，本文技术与Carbon的平均吞吐率分别是软件技术的4.02和2.30倍，这主要是硬件操作延时优势决定的。对于不同的密码算法，软件方式表现出更大的执行性能差异；这是因为软件方式多采用启发式算法，最佳启发效果受测试基准类型、作业大小、核心数目等因素影响。此外，当$\alpha<4$时，软件方式要优于硬件方式，这是核心数目过少

表 3 测试基准说明

测试基准	算法分类	算法特征(bit)	作业数目	平均作业大小(kB)	测试基准	算法分类	算法特征(bit)	作业数目	平均作业大小(kB)	备注
DES	分组	64	1024	54.4	A5	序列	1	1024	41.1	算法特征表示分组/杂凑算法的处理位宽或者序列算法的输出位宽
AES	分组	128	1024	46.3	ZUC	序列	32	1024	50.4	
SM4	分组	512	1024	32.2	RC4	序列	8	1024	47.3	
SHA256	杂凑	512	1024	44.6	RSA	公钥	—	1024	40.3	
SM3	杂凑	512	1024	38.8	SM2	公钥	—	1024	22.8	
SHA-1	杂凑	512	1024	33.2						

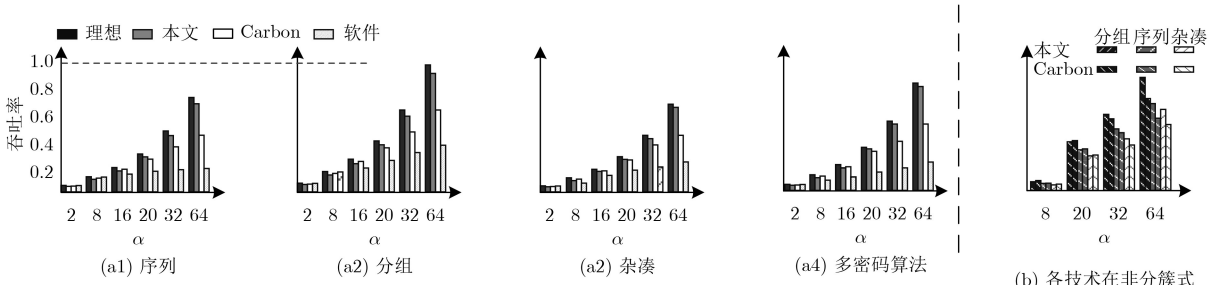


图 5 性能对比图

时，硬件访问开销超过其带来的性能增益所致。

不同硬件技术对比：$\alpha > 20$后，本文技术优于Carbon，且二者性能差逐渐增大(当$\alpha < 20$时，二者表现出相当的性能)。当$\alpha = 64$时，本文技术表现出1.75倍的性能优势。这说明本文技术具有更优的性能可扩展性。原因是α增大导致Carbon的冲突延时和窃取查询时间急剧增大，且Carbon被动的窃取式均衡策略带来了较低的负载均衡频率。

与理想技术的对比:为了探究硬件约束的影响，在上述性能最佳的模型($\alpha = 64$)下，评估本文技术与理想技术在不同队列存储空间下的性能差异。命中开销保持不变，对不命中的情况增加一个210周期的开销成本[5]。对于序列、分组与杂凑算法来说，本文技术与理想技术相比，吞吐率分别降低3～5%，4～7%以及6～9%之间。理想模型中忽略了硬件队列的大小以及访问延时，仍未能带来理想的性能提升的原因是，本文提出的负载均衡引擎包含1个队列预取模块，可以隐藏硬件队列的访问延时。

非分簇式架构下性能对比:在图5(b)中，$m = 1$。实验对比了本文技术与Carbon的吞吐率，当$\alpha = 64$时，本文技术表现出1.36倍的性能优势。进一步，由图5(b)可预测非分簇式架构下，本文技术开始优于Carbon技术的核心数目在20到32之间，这说明本文提出的负载均衡引擎具有可移植性。

4.2.3 延时功耗积评估

众核平台带来高性能也带来了高功耗，实验评估了不同技术(除去理想技术)不同α下的延时功耗积DPP，如图6(a)所示(以$\alpha = 2$时的软件DPP值为参考作标准化处理)。

相同技术的延时功耗积：随着α的增大，即使添加了额外的内核，加之共享存储的争用以及同步开销，DPP仍与α呈现负相关；也就是说虽然α增大带来了额外的功耗开销，但由于显著的性能提升，仍具有较小的DPP；这证明了基于精简处理器核心组体系架构相对于复杂超标量体系架构的优势。

不同技术下的延时功耗积：当$\alpha > 4$左右时，硬件技术表现出优于软件技术的DPP；当$\alpha > 20$左右时，本文所提出的负载均衡引擎开始表现出优于Carbon的DPP。$\alpha = 64$时，Carbon以及基于软件的延时功耗积分别是本文技术的2.45与7.17倍。

4.2.4 资源利用率及负载均衡度分析

根据文献[3]，资源利用率是纯计算时间与总执行时间的比值，而负载均衡度则是所有核心资源利用率的方均根的倒数。3种方案下，不同α下的资源利用率与负载均衡度如图6(b)所示。当$\alpha = 64$时，本文技术的平均资源利用率优于软件作业窃取技术23.01%(最高64.23%)，优于Carbon技术10.2%(最高42.09%)；负载均衡度分别是二者的2.15倍与1.41倍。本文提出的技术大大提高了资源利用率和负载均衡度，且在核心数目增多时有更明显的负载均衡优势。这是因为：(1)本文技术在每个负载均衡周期进行多次作业再分配，这极大提高了负载均

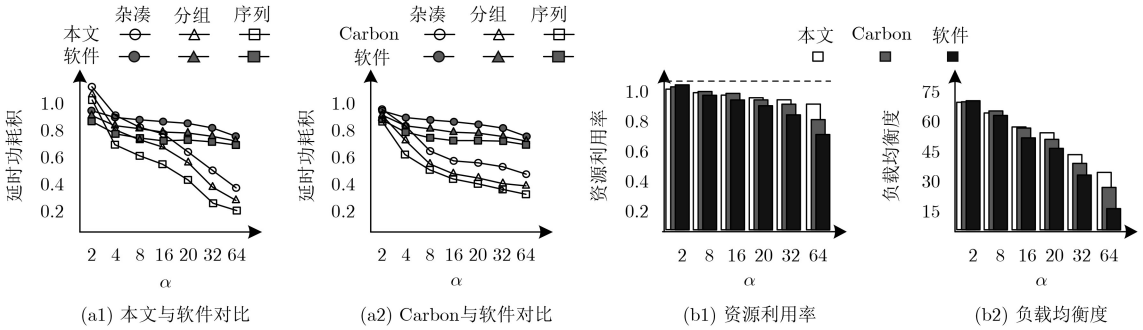


图 6 延时功耗积及资源利用率、负载均衡度对比图

衡效率；(2)基于“作业给予”的负载均衡策略从根本上解决了作业队列技术存在的性能瓶颈——冲突问题。

5 结论

本文基于作业队列技术解决众核密码平台上的负载均衡问题，以提高平台的资源利用率，从而提高密码应用的处理性能。本文建立了关于负载均衡增益率和负载均衡频率的数学模型，并求解了负载均衡操作带来密码平台性能提升的充分条件。基于模型：(1)提出了一种“作业给予”式的负载均衡策略，从根本上解决了“窃取”式负载均衡技术存在的冲突问题；(2)在提高负载均衡增益率与频率的指导思想下，设计了一种基于硬件队列的层次化的负载均衡引擎——“簇内负载均衡阵列-簇间负载均衡微网络”，具有良好的结构可扩展性。实验证明，本文提出的负载均衡引擎在低面积与低功耗开销的前提下，具有理想的性能、延时功耗积、资源利用率以及负载均衡度优势，且具有良好的可移植性和性能可扩展性。

参 考 文 献

[1] KIM C and HUH J. Exploring the design space of fair scheduling supports for asymmetric multicore systems[J]. *IEEE Transactions on Computers*, 2018, 67(8): 1136–1152. doi: 10.1109/TC.2018.2796077.

[2] KIM K W, CHO Y, EO J, *et al*. System-wide time versus density tradeoff in real-time multicore fluid scheduling[J]. *IEEE Transactions on Computers*, 2018, 67(7): 1007–1022. doi: 10.1109/TC.2018.2793919.

[3] LEE J, NICOPOULOS C, LEE H G, *et al*. IsoNet: Hardware-based job queue management for many core architectures[J]. *IEEE Transactions on Very Large Scale Integration Systems*, 2013, 21(6): 1080–1093. doi: 10.1109/TVLSI.2012.2202699.

[4] KUMAR S, HUGHES C J and NGUYEN A. Carbon: Architectural support for fine-grained parallelism on chip multiprocessors[C]. International Symposium on Computer Architecture, California, USA, 2007: 162–173. doi: 10.1145/1250662.1250683.

[5] CHEN J, JUANG P, KO K, *et al*. Hardware-modulated parallelism in chip multiprocessors[J]. *ACM Sigarch Computer Architecture News*, 2005, 33(4): 54–63. doi: 10.1145/1105734.1105742.

[6] LEE J, NICOPOULOS C, LEE Y, *et al*. Hardware-based job queue management for manycore architectures and OpenMP environments[J]. *IEEE International Parallel & Distributed Processing Symposium*, 2011, 21(6): 407–418. doi: 10.1109/IPDPS.2011.47.

[7] 刘宗斌, 马原, 荆继武, 等. SM3哈希算法的硬件实现与研究[J]. 信息网络安全, 2011, 59(9): 191–193. doi: 10.3969/j.issn.1671-1122.2011.09.059.
LIU Zongbin, MA Yuan, JING Jiwu, *et al*. Implementation of SM3 hash function on FPGA[J]. *Information Network Security*, 2011, 59(9): 191–193. doi: 10.3969/j.issn.1671-1122.2011.09.059.

[8] 徐金甫, 杨宇航. SM4算法在粗粒度阵列平台的并行化映射[J]. 电子技术应用, 2017, 43(4): 39–42.
XU Jinfu and YANG Yuhang. Parallel mapping of SM4 algorithm on coarse-grained array platform[J]. *Electronic Technology Application*, 2017, 43(4): 39–42.

[9] DUBEY P. Recognition, mining and synthesis moves computers to the era of tera[J]. *Technology@Intel Magazine*, 2005, 9(2): 1–10.

[10] RATTNER J. Cool codes for hot chips: A quantitative basis for multi-core design[C]. Hot Chips 18 Symposium IEEE, Cupertino, USA, 2016: 1–28. doi: 10.1109/HOTCHIPS.2006.7477745.

[11] AN H, TAURA K, and SPOTTER D. A tool for spotting scheduler-caused delays in task parallel runtime systems[C]. IEEE International Conference on CLUSTER Computing, Hawaii, USA, 2017: 114–125. doi: 10.1109/CLUSTER.2017.82.

[12] KWOK Y K and AHMAD I. Static scheduling algorithms for allocating directed task graphs to multiprocessors[J]. *ACM Computing Surveys*, 1999, 31(4): 406–471. doi: 10.1145/344588.344618.

[13] TITHI J J, MATANI D, MENGHANI G, *et al*. Avoiding locks and atomic instructions in shared-memory parallel BFS using optimistic parallelization[C]. Parallel and Distributed Processing Symposium Workshops & Phd Forum IEEE, Cambridge, UK, 2013: 1628–1637. doi: 10.1109/IPDPSW.2013.241.

[14] MOON S W, REXFORD J, and SHIN K G. Scalable hardware priority queue architectures for high-speed packet switches[J]. *IEEE Transactions on Computers*, 2000, 49(11): 1215–1227. doi: 10.1109/RTTAS.1997.601359.

[15] CHEN Quan, ZHENG Long, and GUO Minyi. Adaptive Demand-aware Work-stealing in Multi-programmed Multi-core Architectures[J]. *Concurrency and Computation: practice & Experience*, 2016, 28(2): 455–471. doi: 10.1002.cpe.3619.

戴紫彬：男，1966年生，教授，博士生导师，研究方向为可重构计算与安全专用芯片设计.

尹安琪：女，1995年生，硕士生，研究方向为可重构计算与信息安全.

曲彤洲：男，1994年生，硕士生，研究方向为可重构计算与信息安全.

南龙梅：女，1981年生，讲师，博士，研究方向为可重构安全芯片设计.