访存与用户行为敏感的 MPSoC 应用映射

王一拙[1],左 琦[2],计卫星[1],王小军[1],石 峰[1]
(1.北京理工大学计算机学院,北京 100081;2.北京市计算中心,北京 100094)

摘 要: 应用映射是 MPSoC 设计中的关键问题,针对多应用负载的 MPSoC,提出一种访存与用户行为敏感的动态映射策略,该策略根据应用的数据访问特征区分热点与非热点应用,并对用户行为进行建模,根据用户行为模型,进一步在运行时区分关键与非关键应用.对每个进入系统的应用,按照应用的热点及关键性分类动态选择在线映射算法,让热点应用围绕存储器布局,非热点应用尽量避免占用存储器附近的资源;对关键应用,最小化应用内通信开销和链路竞争,对非关键应用,最小化应用间通信开销和链路竞争.实验表明,与单纯考虑访存或用户行为的映射策略相比,本文策略能够降低系统整体的通信能耗.

关键词: 多处理器片上系统; 片上网络; 应用映射; 任务映射
中图分类号: TP391 **文献标识码:** A **文章编号:** 0372-2112 (2015)04-0631-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2015.04.002

Memory-Aware and User-Aware Mapping of Applications to MPSoCs

WANG Yi-zhuo[1], ZUO Qi[2], JI Wei-xing[1], WANG Xiao-jun[1], SHI Feng[1]
(1. *School of Computer Science, Beijing Institute of Technology, Beijing* 100081, *China*;
2. *Beijing Computing Center, Beijing* 100094, *China*)

Abstract: Application mapping is one of the key issues in MPSoC design. For MPSoC systems running multiple applications, a memory-aware and user-aware application mapping strategy is proposed. This strategy classifies the applications into hot applications which access large amounts of data and non-hot applications with moderate memory access by offline analysis. Then, the applications are further identified as critical or non-critical application with respect to the user behavior at runtime. For each application, different mapping algorithm is applied according to the above classification. Hot applications are distributed as close as possible to the shared memory. The internal communication cost and contents are minimized for critical applications, and the external communication cost and contents are minimized for non-critical applications. Experimental results show that the proposed strategy saves the overall system communication energy, compared with the single memory-aware or user-aware strategy.

Key words: MPSoC(multiprocessor system-on-chip); NoC(network-on-chip); application mapping; task mapping

1 引言

多处理器片上系统(MPSoC)已成为嵌入式系统设计的主流,随着 MPSoC 集成的处理单元(Processing Element,PE)个数越来越多,其处理能力变得越来越强,促使了多应用负载 MPSoC 的广泛应用,如智能手机、复杂箭载、星载系统等.任务映射和调度在多应用负载的 MPSoC 系统设计中起十分重要的作用,应用通常采用多个任务组成的应用特征图(Application Characterization Graph,ACG)表示[1],如图 1 (*a*),应用映射就是要确定 ACG 中每个节点到 MPSoC 处理单元的分配,如将图 1 (*a*)映射到一个基于 2D-Mesh 片上网络互连的 MPSoC 上后为图 1 (*b*)所示,图中映射策略使节点间总的通信开销最小.

MPSoC 应用映射技术总体上可分为静态映射和动态映射两类[2],静态映射策略在设计时完成任务到 MPSoC 处理单元的分配,因为是离线进行,静态映射可采用复杂的算法,如遗传算法[3]、蚁群算法[4]等,得到优化的映射结果.动态映射策略在运行时决定任务到处理单元的分配,算法要相对简单,否则会对应用程序的总体执行时间造成较大影响.文献[5]对常见动态映射算法进行了介绍和比较,这些算法仅将任务的计算量和通信特征作为任务分配的依据,以控制系统总的通信量为主要目标,没有考虑应用的数据访问特征,更没有考虑用户行为.文献[6,7]提出访存敏感的映射策略,但只针对单应用的静态映射.

收稿日期:2013-12-30;修回日期:2014-06-01;责任编辑:李勇锋
基金项目:国家自然科学基金(No.61300011,No.61300010)

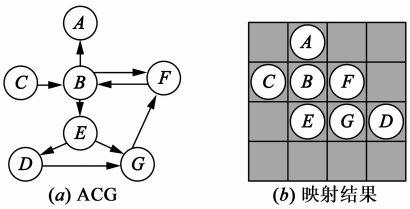

图1 应用映射示例

Chen-Ling Chou 等[8,9]将多应用负载的动态映射归纳为两种主要方式,如图 2 (*a*)和图 2 (*b*),图中灰色区域是系统中已有应用所占用的区域,方法一首先根据当前应用特征图(图 1 (*a*))得到最小化应用内部各节点的通信开销及网络链路竞争的一个不规则区域,然后将这个区域旋转映射到 MPSoC 的空闲节点上,使得剩余节点尽量连通,即剩余节点的总通信开销和应用间链路竞争尽量最小;方法二首先根据 ACG 中的节点个数进行区域选择,保证剩余节点的通信开销和应用间链路竞争最小,然后在选定区域上以最小化应用内部节点通信开销和链路竞争为目标进行映射.总的来说,方法一首先保证当前应用得到最优化的映射结果,其次考虑为未来应用留下优化空间,而方法二首先考虑为未来应用留下优化空间,这样当前应用可能得到次优的映射结果,但对多个应用可能取得较好的总体映射效果.

在考虑访存敏感的基础上,如果能根据应用的关键性,对关键应用采用上述方法一进行映射,非关键应用采用上述方法二进行映射,预期会得到更好的映射效果,这正是本文的研究动机.

本文结合访存敏感和用户行为敏感的 MPSoC 应用映射技术,提出一种混合的动态映射策略.该策略离线分析应用的数据访问特征,将应用区分为热点应用(数据访问量大)和非热点应用,运行时进一步根据用户行为模型(即由用户行为决定的应用特性,如应用运行的频率、时长等)区分关键应用与非关键应用,对每个进入系统的应用,根据其分类采用不同在线映射算法.在

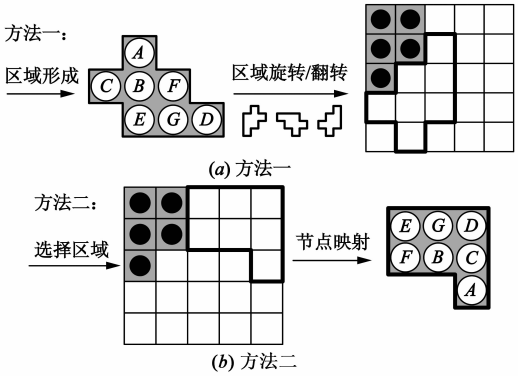

图2 动态应用映射

线映射策略的基本思想是让热点应用围绕存储器,非热点应用尽量避免占用存储器附近的资源;关键应用优先保证当前应用映射效果,非关键应用优先考虑为未来进入系统的应用留下优化空间.

2 系统模型

2.1 目标平台及应用负载模型

片上网络已成为 MPSoC 互连架构的发展趋势,与文献[7,10]等类似,本文面向含有共享存储器节点的基于 2D-Mesh 片上网络互连的 MPSoC,共享存储器位于网络的中心位置,以保证访问它的平均路径最短.

对应用负载采用类似于 ACG 的统一应用特征图(Unified Application Characterization Graph, UACG)[11]来表示,UACG 将数据 buffer 作为根节点,其它节点对应任务(功能划分),节点间边代表任务间通信或任务与数据 buffer 间的数据传输,边上的权重表示通信率,即单位时间从尾节点到头节点任务传输的比特数.

2.2 通信能耗模型

通信能耗分析基于 UACG 的通信率进行,UACG 的通信率既表示了任务之间的通信,也表示了大数据量访存任务与共享存储之间的数据传输.任何应用 Q(边集合表示为 E)在单位时间内总的通信能耗如下:

$$E_{App(Q)} = \sum_{\forall e_{ij} \in E \text{ in } App(Q)} r(e_{ij}) \times E_{bit}(e_{ij}) \tag{1}$$

其中,$r(e_{ij})$ 是应用 Q 某一条边的通信率,$E_{bit}(e_{ij})$代表了从节点 i 到节点 j 所属的部件传输 1 比特所消耗的功率.$E_{bit}(e_{ij})$更为精确的表示如下,包含了路由器上消耗的功率与链路上消耗的功率.

$$E_{bit}(e_{ij}) = E_{Rbit}(e_{ij}) + E_{Link}(e_{ij}) \tag{2}$$

由于本文所针对的 MPSoC 采用 2D-Mesh 拓扑结构,曼哈顿距离(MD)标志了数据包传输所经由的开关数和链路数,是映射策略的主要优化目标.路由器上消耗的能量 $E_{Rbit}(e_{ij})$与链路上消耗的能耗 $E_{Link}(e_{ij})$又可基于曼哈顿距离分别表示如下:

$$\begin{aligned} E_{Rbit}(e_{ij}) &= E_{Rsouce} + (MD-1) \times E_{Rbit} + E_{Rdest} \\ E_{Link}(e_{ij}) &= MD \times E_{Lbit} \end{aligned} \tag{3}$$

其中,$E_{Rsource}$是源节点开关上的能耗,E_{Rdest}是目标节点开关上的能耗,E_{Rbit}是中间经由单个开关的能耗;E_{Lbit}是每两个相邻开关之间链路上的能耗.

在时间间隔 0 ~ T 内,应用 Q 的通信能耗与系统总的通信能耗可通过单位时间能耗 E_{App}和应用的运行时间来计算,分别表示如下:

$$P_{App(Q)} = \left(\sum_{t=1}^{T} \Delta(t) \right) E_{App(Q)} \tag{4}$$

$$P_{Total} = \sum_{t=0}^{T} \sum_{\forall App(i) \text{ ever occured}} E_{App(i)} \tag{5}$$

当 Q 在 $t-1$ 到 t 单位时间间隔内运行时 $\Delta(t)=1$,否则 $\Delta(t)=0$.

2.3 用户行为模型

在多应用负载的 MPSoC 系统中,用户行为,也就是用户对系统中各应用的操作事件,如启动或退出某个应用等,影响着应用在系统中的运行时长和频率等,对用户行为建模就是追踪收集用户操作事件,并根据这一事件集合以及应用本身的通信特征动态决定应用的关键性.本文借鉴文献[8]的方式,用$[t_1, Q, t_2]$表示一个事件,即应用 Q 在时刻 t_1 之后进入系统,并在运行一段时间后于 t_2 时刻之前离开系统.从应用的通信强度与活跃程度两个角度判定应用的关键性,采用通信率比例表示通信强度,累积通信能耗比例表示活跃程度,下面给出这两个参数的定义:

参数 1　通信率比例

$$\alpha = \frac{\sum\limits_{\forall e_{ij}\in E \text{ in } App(Q)} r(e_{ij})}{\sum\limits_{\forall App(Q) \text{ ever occured}} \sum\limits_{\forall e_{ij}\in E \text{ in } App(Q)} r(e_{ij})} \tag{6}$$

通信率比例反映的是应用 Q 的通信率在系统整体通信率占用的比重,从而反映了应用 Q 的通信强度.

参数 2　累积通信能耗比例

$$\beta = \frac{P_{App(Q)}}{P_{\text{Total}}} \tag{7}$$

累积通信能耗比例反映的是应用 Q 的活跃程度.从式(4)~(7)可知,用户操作事件的集合影响着 α 和 β 的计算,用户行为敏感的映射策略设置两个阈值 α_{th}和 β_{th},当应用 Q 到达系统进行映射时,计算 Q 的 α 和 β 值,如果 $\alpha>\alpha_{\text{th}}$或 $\beta>\beta_{\text{th}}$,则该应用判定为关键应用,否则判定为非关键应用.

在系统的初始时刻,由于没有足够的用户操作事件信息,α 和 β 无法计算或计算结果无实用意义,因此在初始时刻不进行关键性判断,始终选用上文中的方法二进行映射.文献[8]取 α_{th}和 β_{th}为经验值 0.8 和 0.7,文献[9]给出一种简单的机器学习过程,动态决定 α_{th}和 β_{th}的取值,本文的目的在于说明访存与用户行为敏感相结合的映射效果,而不在于用户行为模型的改进,因此同文献[8],采用固定阈值.

3 映射策略与问题分割

3.1 映射策略框架

本文映射策略框架如图 3 所示,包含三个步骤:第一步提取离线应用特征,分析待运行应用,将其表示为 UACG;第二步是初始映射,该步中映射遵循热点应用围绕共享存储器的原则,非热点应用避开存储器,并以上文方法二进行映射,在进行应用映射的同时启动对用户行为的追踪;第三步根据用户行为模型与系统当前配置决定新进入系统的应用映射,这一步结合热点及关键性判断在不同的映射算法中做出合理选择.

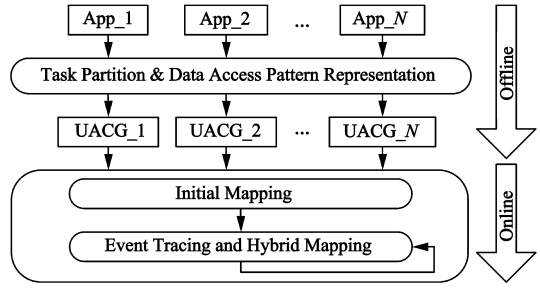

图3　访存与用户行为敏感的映射策略框架

图 4 是本文映射策略在线部分的算法流程,到达系统的应用已经由离线步骤转化为应用的 UACG 表示.当应用启动事件触发时,先判断系统资源是否满足当前应用的需求,如果资源不足则拒绝该应用启动事件,具体实现为将当前事件放入等待队列中,等系统可用资源充足时再执行.如果系统资源满足应用需求,开始进行应用映射,映射算法首先判断当前应用类型为热点应用还是非热点应用,进一步,又根据其关键性和用户行为模型区分关键应用与非关键应用,选择不同的映射方法(方法 1~4),每种方法分别包含两个子问题(P1,2;P3,4;P5,6;P7,8).方法 1,3 首先根据应用自身特征形成所需区域(P1,P5),之后将这一区域旋转映射到系统中的适当位置(P2,P6);方法 2,4 首先在系统中选择一块能容纳当前应用的区域(P3,P7),随后将应用中的任务逐一分配到选定区域内的资源节点上(P4,P8).方法 1,3 针对关键应用,以最小化应用内部链路竞争和通信开销为目标,保证了关键应用的优化映射;方法 2,4 针对非关键应用,以最小化应用间链路竞争和通信开销为目标,为后继应用尽量提供优化配置的空间.

3.2 在线映射的问题分割

为描述方便,先给出如下定义:

(1) $MD(s_i=(x_i,y_i), s_j=(x_j,y_j))$:节点 s_i 与 s_j 之间的曼哈顿距离,其中 x_i, y_i, x_j, y_j 表示在 mesh 系统中的 x 坐标与 y 坐标,$MD(s_i,s_j)=|x_i-x_j|+|y_i-y_j|$;

(2) $ED(s_i=(x_i,y_i), s_j=(x_j,y_j))$:节点 s_i 与 s_j 之间的欧几里得距离,$ED(s_i,s_j)=(|x_i-x_j|^2+|y_i-y_j|^2)^{1/2}$;

(3) R:一个包含若干节点的区域,该区域可以是连续的也可以是非连续的;

(4) $L(R)$:区域 R 内部所有节点两两之间曼哈顿距离的总和.

MD 用来衡量节点之间的距离,ED 用来判断节点

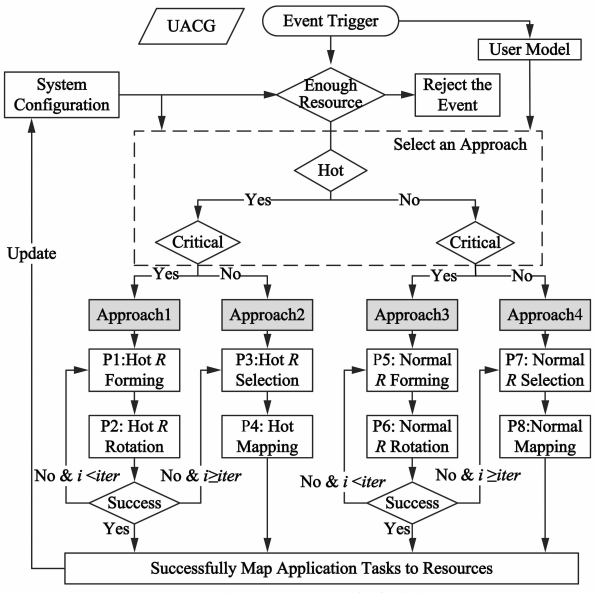

图4 在线分配策略算法流程图

到中心节点的远近.用 R' 表示当前系统中可获取的空余资源所构成的区域,R 表示为新到来的应用分配的资源所占区域,图 4 中的 8 个子问题可描述如下.

关键热点应用区域形成子问题(P1):给定应用的 UACG,找到适合应用的区域 R 和任务在 R 中对应的位置分布 G,使得:

(1)共享数据 buffer 位于区域 R 中;

(2)每个 PE 上只分配一个任务;

(3)最小化 R 中所有链路的通信开销总和.

关键热点应用区域旋转子问题(P2):给定已经形成的区域 R(由 P1 生成)和系统当前配置 $Conf$,找到一种将 R 置于 R' 内的方法且最小化 $L(R)+L(R'-R)$.

非关键热点应用区域选择子问题(P3):给定应用的 UACG(V,E)与系统的当前配置,找到 R' 中的一个区域 R,使得:

(1) R 中的资源数目等于 UACG 的资源需求,即 $\|R\|=\|V\|$;

(2)共享存储器在 R 内,即 $Loc(mem)\in R$;

(3) R 中的节点以最小化 MD 为标准围绕共享存储器,即 $\forall pos\in R,\min\{MD(pos,mem)\}$;

(4) $\min\{L(R)+L(R'-R)\}$.(注:根据已有研究[1],最小化外部冲突(应用间链路竞争)的标准是选择 $L(R)+L(R'-R)$最小的区域.)

非关键热点应用节点映射子问题(P4):给定应用的 UACG 与已经选择好的区域 R(由 P3 生成).找到该应用在区域 R 内的一个映射,使得:

(1)UACG 中的根节点映射到共享存储器上;

(2)任务各自的资源两两不相同;

(3)以存储约束为第一要素选择任务围绕共享存储器;

(4)最小化应用内冲突(任务间链路竞争).

关键非热点应用区域形成子问题(P5):给定应用的 UACG,找到适合应用的区域 R 和任务在 R 中对应的位置分布 G,使得:

(1)每个 PE 上只分配一个任务;

(2)最小化 R 中所有链路的通信开销总和;

(3)存在多种可能时选择 $L(R)$最小的形状.

关键非热点应用区域旋转子问题(P6):给定区域 R(由 P5 生成)和系统当前配置,找到一种将 R 置于 R' 内的方法且最小化 $L(R)+L(R'-R)$.

非关键非热点应用区域选择子问题(P7):给定应用的 UACG 与系统的当前配置,找到 R' 中的一个区域 R,使得:

(1) R 中的资源数目等于 UACG 的资源需求;

(2)共享存储器不在 R 内;

(3) $\min\{L(R)+L(R'-R)\}$;

(4)优先选择远离共享存储器的资源.

非关键非热点应用节点映射子问题(P8):给定应用的 UACG 与已经选择好的区域 R(由 P7 生成),找到应用在 R 内的一个映射,使得:

(1)UACG 每个节点都分配一个资源,节点不同资源不同;

(2)最小化应用内冲突.

4 相关算法

非关键应用的映射算法(P3,P4,P7,P8)与文献[11]所使用的算法相同,文献[11]只考虑访存特征,区分热点与非热点应用,给出了热点与非热点应用各自的区域选择与节点映射算法,相当于所有应用都按照本文非关键应用对待.鉴于篇幅限制,下面仅介绍本文关键应用的相关算法.

4.1 关键热点应用的相关算法

关键热点应用的映射算法包含区域形成(P1)与区域旋转(P2)两个子算法.其中 P1 根据热点应用的 UACG 生成围绕 buffer 的最优区域,P2 将该区域映射到系统上,找到容纳该最优区域的空闲资源.

区域形成(P1)如算法 1 所示,输入为当前应用的 UACG,输出为形成的区域 R 以及 UACG 的各节点在 R 中的对应位置 G.首先确定 buffer 为区域中心,即 $G[b]=g_{0,0}$,b 是 buffer 节点,也就是 UACG 的根节点,$g_{0,0}$表示区域中心位置;其次遍历与 buffer 相连的节点,将这些节点以最小化总的通信量和链路冲突为约束布置在 buffer 周围,即算法 1 中的第一个 while 循环,其中 V' 是 buffer 相邻的节点集合,$V_{x,y}$是 $g_{x,y}$到 $g_{0,0}$的路由路径上

的节点集合;最后遍历剩余节点,将各剩余节点安排到与 UACG 中同其相连的已分配节点 *MD* 最小的位置上,若有多个 *MD* 最小的位置,则进一步选择与中心点 *ED* 最小的位置,即算法 1 中的第二个 while 循环,其中 u' 是任一个已分配节点.

算法 1　关键热点应用的区域形成

输入:UACG(V, E)
输出:R, G
for each $u \in V$ do
　$flag[u] = 0$;
end for
$R = \Phi$;
$flag[b] = 1$;
$G[b] = g_{0,0}$;
$R = R \cup \{G[b]\}$
while $\sum_{\forall u \in V'} |flag[u]| < |V'|$ do
　choose $u \in V'$ with $flag[u] == 0$ and $r(e_{ub})$ is maximized.
　$flag[u] = 1$;
　choose available location $g_{x,y}$ such that $\sum_{\forall v \in V' and\ flag[v]==1} (MD(g_{x,y}, G(v)) \times r(e_{uv}))$ is minimized
　and then $\sum_{\forall v \in V_{x,y}} (MD(g_{0,0}, G(v)) \times r(e_{bv}))$ is minimized.
　$G[u] = g_{x,y}$; $R = R \cup \{g_{x,y}\}$;
end while
while $\sum_{\forall u \in (V-V')} |flag[u]| < |V - V'|$ do
　choose $u \in (V - V')$ with $flag[u] == 0$ and $r(e_{uu'})$ is maximized.
　$flag[u] = 1$;
　choose available location $g_{x,y}$ such that $MD(g_{x,y}, G(u'))$ is minimized
　and then $ED(g_{x,y}, g_{0,0})$ is minimized.
　$G[u] = g_{x,y}$;　$R = R \cup \{g_{x,y}\}$;
end while

区域旋转(P2)如算法 2 所示,首先将区域中心 $g_{0,0}$ 固定映射到当前系统配置 *Conf* 的中心位置,即共享存储器上;然后将 P1 形成的区域 R 进行 90,180,270 度旋转,以及上下左右翻转组成区域集合 *Rotation_Set*,集合中的每个区域都用边框矩形表示,边框矩形中区域所在点标识为 1,其它点标识为 0;接着对旋转和翻转形成的每个区域调用 *Match* 函数判断当前的系统配置能否容纳该最区域,*Conf* 也用 0、1 标识的矩形表示,*Match* 函数只需判断 R_{rec} 中标识为 1 的点在 *Conf* 中对应位置是否标识为 1,如果是的话则不匹配;最后,如果存在多种可能的匹配,选择最小化资源碎片的匹配方式.另外,如图 4,设定一个阈值,尚未达到阈值时若 MPSoC 不能容纳最佳区域,则形成另一种最佳区域;如果到达阈值时 MPSoC 仍不能容纳最佳区域,则只能回退到最小化全局通信的方式进行区域选择(P3)与应用映射(P4).

算法 2　关键热点应用的区域旋转

输入:*Conf*, R
输出:*MPG*(R -> *Conf*)
Anchor $g_{0,0}$ to the center of *Conf*
Construct *Rotation_Set*
for each R_{rec} in *Rotation_Set* do
　if *Match*(R_{rec}, *Conf*) then
　　Match_Set = *Match_Set* ∪ *MPG*(R_{rec} -> *Conf*)
　end if
end for
choose *mpg* in *Match_Set* such that *Fragment*(*mpg*) is minimized.
return *mpg*

4.2　关键非热点应用的相关算法

对关键非热点应用,区域形成算法(P5)计算应用所包含的任务能够形成的最佳区域,如算法 3 所示,首先将与其它任务通信总量最大的节点作为区域中心,然后以 *MD* 与 *ED* 双重约束选择应用中其它任务的位置,u' 表示任意已分配节点,当多个节点与已分配节点有相同 *MD*、*ED* 时,优先选择最小化 $L(R)$的位置.

算法 3　关键非热点应用的区域形成

输入:UACG(V, E)
输出:R, G
for each $u \in V$ do
　$flag[u] = 0$;
end for
$R = \Phi$;
choose u such that $\sum_{\forall v \in V} r(e_{uv})$ is maximized.
$flag[u] = 1$;
$G[u] = g_{0,0}$;
$R = R \cup \{G[u]\}$
while $\sum_{\forall u \in V} |flag[u]| < |V|$ do
　choose $u \in V$ with $flag[u] == 0$ and $r(e_{uu'})$ is maximized.
　$flag[u] = 1$;
　choose available location $g_{x,y}$ such that $MD(g_{x,y}, G(u'))$ is minimized
　and then $ED(g_{x,y}, G(u'))$ is minimized
and then $L(R \cup \{g_{x,y}\})$is minimized.
　$G[u] = g_{x,y}$;　$R = R \cup \{g_{x,y}\}$;
end while

区域旋转(P6)在当前系统中为 P5 所形成的区域选择具体映射位置,如算法 4 所示,首先对输入 R 旋转与翻转形成集合 *Rotation_Set*,然后按照离中心共享存储器曼哈顿距离的递减顺序查找可能的映射位置,即算法 4 中的 while 循环,每执行一次后调用 *Shrink* 函数将 *Conf* 网格向中心缩进一格,直到映射目标网格区域大小不大于待映射的区域 R 的大小. *Match_margin* 函数按顺时针方向沿 *Conf* 的网格边沿逐格移动 R_{rec},判断

R_{rec}是否能映射到相应位置,当有多个可能的映射时选择最小化资源碎片的映射.

算法 4 关键非热点应用的区域旋转

输入:*Conf*, *R*
输出:*MPG*(*R* -> *Conf*)
Construct *Rotation_Set*
while *Size* (*Conf*) ≥ *Size* (∀ R_{rec} ∈ *Rotation_Set*) do
  for each R_{rec} in *Rotation_Set* do
    if *Match_margin*(R_{rec}, *Conf*) then
      Match_Set = *Match_Set* ∪ *MPG*(R_{rec} -> *Conf*)
      break;
      end if
    end for
    if *Match_Set* ! = Ø then
      choose *mpg* in *Match_Set* such that *Fragment*(*mpg*) is minimized
      return *mpg*;
    end if
    Shrink (*Conf*);
end while

5 实验与分析

5.1 实验方法

实验比较如下 MPSoC 动态应用映射技术:

(1)访存与用户行为敏感的映射策略,即本文技术(简记为 MUA);

(2)仅用户行为敏感的映射策略[8](简记为 UA);

(3)访存敏感的映射策略一,即图 2(*a*)方法基础上考虑访存敏感,(简记为 MA-Ⅰ);

(4)访存敏感的映射策略二[11],即图 2(*b*)方法基础上考虑访存敏感,(简记为 MA-Ⅱ).

UA 不考虑共享存储器的位置,具体算法参照文献[8]实现,输入为应用的 ACG 表示. MA-Ⅰ考虑访存敏感,并将所有应用都当做关键应用对待,具体实现即图 4 中的 Approach1 (P1, P2)和 Approach3 (P5, P6)两个分支;MA-Ⅱ考虑访存敏感,所有应用都当做非关键应用对待,具体实现即图 4 中的 Approach2 (P3, P4)和 Approach4 (P7, P8)两个分支. MUA、MA-Ⅰ和 MA-Ⅱ都将应用表示为 UACG 作为算法输入.

实验分为两个部分:一是根据本文第 2 节建立系统通信能耗的理论分析模型,对合成应用负载进行分析计算,比较不同映射策略下的通信能耗;二是选取真实应用负载,用 Noxim 模拟器测试不同映射策略下的通信能耗.

5.2 合成负载下的评估

首先采用 TGFF[12] 工具生成 10 个应用负载的 UACG,其中包含 3 个热点应用、7 个非热点应用. 非热点应用分别包含 3 到 10 个任务节点,热点应用分别包含 6 个、8 个与 10 个任务节点,其中访存节点数分别为 4 个、5 个与 5 个. 其次生成 200 个用户操作事件(见 2.3 节),每个事件[t_1, Q, t_2]表示应用 Q 在系统中从时刻 t_1 到时刻 t_2 运行,前 50 个事件的 Q 从上述 10 个应用中随机选取,之后提高两个热点应用和两个非热点应用的出现频率,使其成为关键应用.(注:是否关键应用在运行时动态决定,这里只是提高部分应用的活跃程度,使其判定为关键应用的几率变大.)

实验模拟的 MPSoC 采用 7 × 7 节点的 2D-Mesh 结构,中心节点设为多端口共享存储器. MUA 中 α_{th}和 β_{th}参考文献[8]取经验值 0.8 和 0.7. 每个时刻 t,计算 $t-1$ 到 t 时刻的系统总体通信能耗. 不同映射策略下的能耗对比如图 5 所示,水平轴是整型的时刻单元,垂直轴是本文策略与其它策略的通信能耗比值.

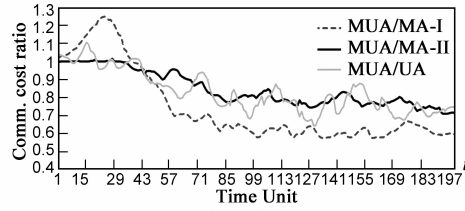

图5 系统通信能耗对比

从图 5 可见,与 MA-Ⅰ相比,初始时刻由于系统可分配资源充足,MA-Ⅰ先形成应用的最佳分配区域,该区域此时总能映射到系统上连续的空闲节点,而 MUA 在初始时刻采用先选择近似凸形区域再进行节点映射的方式,往往得不到应用的最佳布局,因此初始时刻 MUA 的通信能耗高于 MA-Ⅰ,在系统运行一段时间后,MA-Ⅰ造成资源碎片较多,对后继应用的映射非常不利,相比 MUA,时刻 50 之后 MA-Ⅰ的能耗高出约 36.4%;与 MA-Ⅱ相比,由于初始时刻(t = 20 之前)MUA 采用与 MA-Ⅱ相同的映射算法(见 2.3 节),因此两者比值为 1,之后随着对关键应用的区分,MUA 逐渐优于 MA-Ⅱ并保持相对稳定的优势,时刻 50 之后 MUA 对比 MA-Ⅱ的平均能耗节约为 20%;与 UA 相比,不同时刻的能耗比值变化较大,但总体来看,MUA 优于 UA 约 22.3%(时刻 50 – 200 的均值),说明考虑应用的访存特征能够明显降低系统通信能耗.

5.3 真实负载下的评估

为评估本文技术,我们对 Noxim[13]模拟器进行了扩展,将动态映射算法模块集成到 Noxim 模拟器中,由映射算法模块为每个 PE 根据当前应用的 UACG 和映射结果产生数据包. 实验中微片大小设置为 128 比特,采用 XY 维序路由算法和随机输出端口选择策略,模拟器所需的能耗参数采用 Orion 2.0[14] 获得,所有参数均在

45nm 工艺,0.8V 电源电压,3GHz 主时钟频率下得到.选用的真实应用见表 1,根据访存特征将其中 4 个归为热点应用,5 个归为非热点应用,所有应用都事先进行了任务划分,并通过 Profiling 得到各任务间通信量,构造 UACG 作为 Noxim 模拟实验输入.实验模拟的 MPSoC 结构、用户操作事件的生成、阈值 α_{th}和 β_{th}的设置均与 5.2 节相同.

表 1　真实应用负载

应用	描述	任务数	热点
DenseMM	分块并行矩阵乘法(2000 * 2000 double matrix)	12	√
MSort	对 800M 大小整数数据并行归并排序	8	√
JPEG	JPEG 解码 1628 * 1024 图像	9	√
MP3	MP3 解码 1.8M 音频文件	7	√
NQueens	n 皇后问题(n = 12)	12	
SparseLU	稀疏矩阵 LU 分解	8	
Fibonacci	递归计算第 n 个斐波那契数(n = 40)	10	
Wavelet	2D 小波变换	8	
Bitcount	处理器位操作性能测试程序,来自 MiBench[15].	6	

图 6 显示了实验测得的不同应用通信能耗以及系统总的通信能耗,即 2.2 节的 $P_{App(Q)}$和 P_{Total}.实验结果以本文映射策略为基准归一化,图中可见:(1)采用本文映射策略系统总体通信能耗小于其它三种映射策略,能耗节约的比例小于 5.2 节理论模型的计算值;(2)热点应用采用访存敏感的映射策略(MUA,MA-Ⅰ,MA-Ⅱ)通信能耗均小于非访存敏感的映射策略(UA),平均来看 MUA 的能耗节约最大;(3)某些非热点应用采用 UA 的能耗小于 MA-Ⅰ或 MA-Ⅱ,这是因为 UA 不考虑存储器位置的限制,非热点应用相比其它策略更容易得到优化的映射结果.

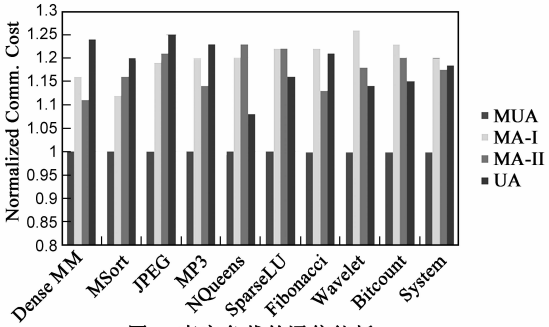

图6　真实负载的通信能耗

6　总结

本文设计了一种动态 MPSoC 应用映射策略,该策略面向片上网络互连的 MPSoC,对包含热点应用(含有大量访存操作任务)的多应用负载进行动态映射,并考虑用户行为习惯形成的应用关键性区分,在运行时选择不同映射算法.实验表明,这种混合了热点与关键性判定的映射策略比已有的用户行为敏感映射策略降低系统通信能耗约 22.3%,比单纯访存敏感的映射策略降低通信能耗约 20%.

参考文献

[1] J Hu, R Marculescu. Energy and performance-aware mapping for regular NoC architectures[J]. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems, 2005, 24(4): 551 – 562.

[2] P K Sahu, S Chattopadhyay. A survey on application mapping strategies for Network-on-Chip design[J]. Journal of System Architecture, 2013, 59(1): 60 – 76.

[3] T Lei, S Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture[A]. Proc of Euromicro Symp on Digital System Design[C]. Washington: IEEE Computer Society, 2003. 180 – 187.

[4] 易伟,王佳文,潘红兵,等.基于蚁群混沌遗传算法的片上网络映射[J].电子学报,2011,39(8):1833 – 1836.
Yi Wei, Wang Jiawen, Pan Hongbing, et al. Ant colony chaos genetic algorithm for mapping task graphs to a network on chip [J] Acta Electronica Sinica, 2011, 39(8): 1833 – 1836. (in Chinese)

[5] E Carvalho, N Calazans, F Moraes. Dynamic task mapping for MPSoCs[J]. IEEE Design and Test of Computers, 2010, 27(5): 26 – 35.

[6] H Orsila, T Kangas, E Salminen, et al. Automated memory-aware application distribution for multi-processor system-on-chips[J]. Journal of Systems Architecture, 2007, 53 (11): 795 – 815.

[7] S Lee, Y Yoon, S Hwang. Communication-aware task assignment algorithm for MPSoC using shared memory[J]. Journal of Systems Architecture, 2010, 56(7): 233 – 241.

[8] C Chou, R Marculescu. User-aware dynamic task allocation in networks-on-chip[A]. Proc of the Design, Automation and Test in Europe[C]. New York: ACM, 2008. 1232 – 1237.

[9] C Chou, R Marculescu. Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip [J]. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems, 2010, 29(1): 78 – 91.

[10] J Lee, K Choi. Memory-aware mapping and scheduling of tasks and communications on many-core SoC[A]. Proc of the 17th Asia and South Pacific Design Automation Conf[C]. Piscataway, NJ: IEEE, 2012. 419 – 424.

[11] 王一拙,左琦,计卫星,等.访存敏感的增量式 MPSoC 应用映射[J].计算机研究与发展,2015,52(1):1 – 9.

Wang Yizhuo, Zuo Qi, Ji Weixing, et al. Memory-Aware Incremental Mapping of Applications to MPSoCs[J]. Journal of Computer Research and Development, 2015, 52(1): 1 – 9. (in Chinese)

[12] R P Dick, D L Rhodes, W Wolf. TGFF: Task graphs for free [A]. Proc of the 6th Int Workshop on Hardware/Software Codesign [C]. Washington, DC: IEEE Computer Society, 1998. 97 – 101.

[13] F Fazzino, M Palesi, D Patti. Noxim: Network-on-chip simulator[OL]. http://noxim.sourceforge.net, 2010 – 10 – 16.

[14] A B Kahng, B Li, L S Peh, et al. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration[A]. Proc of the Conf on Design, Automation and Test in Europe[C]. Belgium: European Design and Automation Association, 2009. 423 – 428.

[15] M R Guthaus, J S Ringenberg, et a1. Mibench: A free commercially representative embedded benchmark suite[A]. Proc of the 4th IEEE Int'l Workshop on Workload Characterization [C]. Piscataway, NJ: IEEE, 2001. 3 – 14.

作者简介

王一拙 男. 1979 年 2 月出生, 陕西西安人. 现为北京理工大学计算机学院讲师, 主要研究方向: 并行程序设计、计算机体系结构、嵌入式系统.

E-mail: frankwyz@bit.edu.cn

左 琦 女. 1983 年 2 月出生, 河北张家口人. 现为北京市计算中心工程师, 主要从事体系结构、并行计算、云计算等方面的研究工作.

E-mail: zuoqi@bcc.ac.cn