一种负载感知的异构MPSoC任务调度算法

谢 盈[1,3,5]，吴尽昭[2]，丁旭阳[4]，张 晖[1,3]

(1. 中国科学院成都计算机应用研究所 成都 610041；2. 广西民族大学广西混杂计算与集成电路设计分析重点实验室 南宁 530006；
3. 中国科学院大学 北京 石景山区 100049；4. 电子科技大学计算机科学与工程学院 成都 611731；
5. 西南民族大学计算机科学与技术学院 成都 610041)

【摘要】处理器核的异构性、运行时负载和任务间依赖关系，是影响异构MPSoC任务调度算法性能的关键因素。该文提出了一种负载感知的异构MPSoC任务调度算法，在满足任务间依赖关系的前提下，根据计算开销和通信负载将待调度任务集划分为任务子集。在考虑处理器核负载状态的基础上，通过赋权二部图最大权匹配，将任务子集调度到适载的处理器核上运行，提高了待调度任务集总执行效率。仿真实验结果表明，该算法有效降低了任务集的调度长度，提高了处理器核的利用率。

关 键 词 异构MPSoC； 负载感知； 任务调度； 任务划分
中图分类号 TP301 文献标志码 A doi:10.3969/j.issn.1001-0548.2017.06.017

A Load-Aware Task Scheduling Algorithm on Heterogeneous MPSoC

XIE Ying[1,3,5], WU Jin-zhao[2], DING Xu-yang[4], and ZHANG Hui[1,3]

(1. Chengdu Institute of Computer Application, Chinese Academy of Sciences Chengdu 610041;
2. Guangxi Key Laboratory of Hybrid Computational and IC Design Analysis, Guangxi University for Nationalities Nanning 530006;
3. University of Chinese Academy of Sciences Shijingshan Beijing 100049;
4. School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 611731;
5. School of Computer Science and Technology, Southwest University for Nationalities Chengdu 610041)

Abstract The performance of task scheduling algorithm on heterogeneous MPSoC is affected by heterogeneous cores, run-time load and tasks dependencies. A novel load-aware task scheduling algorithm is proposed on heterogeneous MPSoC, which divides task-set into task-subsets based on tasks dependencies, computation overhead and communication overhead. In considering the core's load state, task-subsets are dispatched to appropriate cores by maximum weight matching of weighted bipartite graph, which improves the overall efficiency of task-set. Simulation results show that the proposed algorithm can reduce the length of task-set scheduling and improve the utilization of cores.

Key words heterogeneous MPSoC; load-aware; tasks dispatch; tasks divide

片上多核处理器MPSoC(multi-processor system-on-chip)把多个处理核心集成到一个芯片中，通过这些处理核心的并行工作提高系统性能，以满足日益增长的功能及性能需求[1]。其中，异构MPSoC中的处理器核心具有不同结构、地位、功耗及运算能力，大幅度增强了信息处理能力。在实际应用中，任务调度算法的优劣直接影响异构MPSoC系统的利用率[2]，是被广泛关注的研究热点。

目前，针对复杂任务的调度技术主要有随机搜索调度技术和启发式调度技术[3-4]。随机搜索调度技术如遗传算法[5-6]、模拟退火算法[7]等，随着任务数和核数量的增加，搜索开销和计算复杂度呈非线性增长，收敛速度急剧下降。而启发式调度技术因设计简单且能够获得较好的次优解，获得更多研究者的青睐，涌现出大量研究成果，如：对任务进行分簇的边消除算法[8]；为了减小通信依赖而对前驱任务进行复制的任务复制算法[9]；根据任务属性构造任务列表以依次调度任务的最早截止时间优先算法[10]、关键路径算法[11]、优先级调度算法[12]等。但是在任务间依赖关系复杂、异构处理器核性能差异较大的情况下，以上算法灵活性较差，且存在处理器负载不均的问题。为此，研究人员在考虑核异构

收稿日期：2016 – 04 – 21；修回日期：2016 – 07 – 15
基金项目：国家自然科学基金(11371003，11461006)；广西自然科学基金(2012GXNSFGA060003)；中央高校基本科研业务费(2015NZYQN28).
作者简介：谢盈(1984 –)，女，博士生，主要从事嵌入式系统、形式化方法方面的研究.

性的基础上，应用启发式调度技术以获得更好的调度性能。HEFT(heterogeneous earliest-finish-time)算法[13]将EDF算法应用于异构环境，是异构环境下很多其他调度模型的基础，但缺乏处理系统过载的能力。文献[14]提出的PLTSF(probably lag time shortest first)算法，通过复制fork节点将具有复杂依赖关系的任务转化为多个相互独立的并行任务树，再动态地计算任务树优先级，根据优先级将任务分配到处理器核上，但该算法每次只选择一个任务进行映射，效率较低。

本文提出了一种具备负载感知能力的异构MPSoC任务调度算法LATS(load-aware task scheduling)，依据任务计算开销和任务间通信负载，将具有依赖关系的任务集划分为并行任务子集，在考虑处理器核负载状态的前提下，根据赋权二部图的最大权匹配将多个任务子集优化调度到适载处理器核上，提高了待调度任务集的总执行效率。仿真实验及结果表明，本文提出的算法在负载感知的基础上，有效降低了任务集的调度长度，提高了处理器核利用率。

1　系统模型

定义 1　异构MPSoC环境是一个4元组 $E=(\text{Core},\text{Speed},\text{Cap},\text{Que})$。

1) $\text{Core}=\{\text{Core}_0,\text{Core}_1,\cdots,\text{Core}_{N\text{-}1}\}$ 是具有 N 个处理器核的片上系统，$N\geqslant 2$。

2) $\text{Speed}=\{S_0,S_1,\cdots,S_{N-1}\}$ 是处理器核运算速率的集合，处理器核的运算速率是由主频、流水线、缓存等因素共同决定，为简化模型，本文使用 S_i 刻画第 i 个核的运算速率。

3) $\text{Cap}=\{C_0,C_1,\cdots,C_{N-1}\}$ 是处理器核存储能力的集合，C_i 表示第 i 个核总存储容量。

4) $\text{Que}=\{Q_0,Q_1,\cdots,Q_{N-1}\}$ 是处理器核维护的等待队列的集合，Q_i 表示第 i 个核所维护的等待队列，$|Q_i|$ 表示 Core_i 上排队等待执行的任务数。

定义 2　待调度的任务集可表示为DAG图，$DAG=((T,R),\boldsymbol{C},\text{TS})$。

1) $T=\{T_0,T_1,\cdots,T_{n-1}\}$ 是任务节点集合，$n\geqslant 1$。T_i 对应任务集中的一个任务。

2) $R=\{r_0,r_1,\cdots,r_{n-1}\}$ 是任务权重集合，r_i 表示任务 T_i 的计算开销，$r_i\in R^+$。

3) $\boldsymbol{C}$ 是任务间通信权重矩阵，$c_{i,j}$ 表示任务 T_i 到 T_j 的通信负载，$c_{i,j}\geqslant 0$；若 $c_{i,j}=0$ 表示任务 T_i 与 T_j 没有依赖关系。

4) TS为虚拟起始节点，TS的计算开销为0，TS到其直接后续的通信负载为0。

如图1所示，该DAG图表示了 n 个任务的计算开销，任务间依赖关系和通信负载。后续任务需要使用其所有直接前驱任务的输出结果或占用的系统资源，因此，后续任务要等其所有直接前驱任务都执行结束后，才能开始被调度执行。

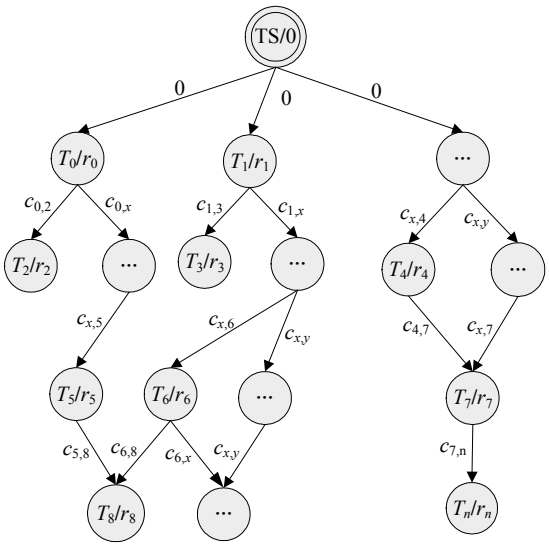

图1　任务集DAG图示

2　负载感知异构MPSoC任务调度算法

2.1　任务子集划分

为了充分利用MPSoC丰富的计算资源，LATS算法在任务子集划分阶段以尽量降低任务间通信负载为目标，将含有n个任务的任务集划分为m个并行任务子集。

任务直接前驱所在的集合称为前驱任务子集，任务的最大前驱开销 $\text{MaxPredCost}(T_i)$ 是当前任务 T_i 的所有前驱任务子集的开销与通信负载之和的最大值。

$$\text{MaxPredCost}(T_i)=\begin{cases}\max\left(\left(\sum_{k=0}^{|\text{Task}_j|-1} r_k+c_{j,i}\right),\left(\sum_{k=0}^{|\text{Task}_j|-1} r_k-r_i\right)\right) & T_i\in\text{Task}_i\\ \max\left(\left(\sum_{k=0}^{|\text{Task}_j|-1} r_k+c_{j,i}\right)\right) & T_i\notin\text{Task}_i\end{cases}$$
$$\forall T_j,(T_j\in\text{pred}(T_i))\wedge(T_j\in\text{Task}_j)\wedge(T_i\notin\text{Task}_j) \tag{1}$$

式中，$T_j\in\text{pred}(T_i)$ 表示任务 T_j 是 T_i 的直接前驱；Task_j 是 T_j 所在的前驱任务子集；$T_i\in\text{Task}_i$ 表示 T_i 已经被划分到任务子集 Task_i 中。

LATS 任务子集划分算法如图 2 所示。

1) 按任务节点索引值递增的顺序遍历所有节点。任务节点索引值定义为：

$$\mathrm{Index}(T_i)=\begin{cases}0 & \mathrm{pred}(T_i)=\varnothing\\ 1+\max(\mathrm{Index}(T_j)) & T_j\in\mathrm{pred}(T_i)\end{cases}$$

2) 如果 T_i 的直接前驱为 TS，则 T_i 单独形成任务子集 Task_i。

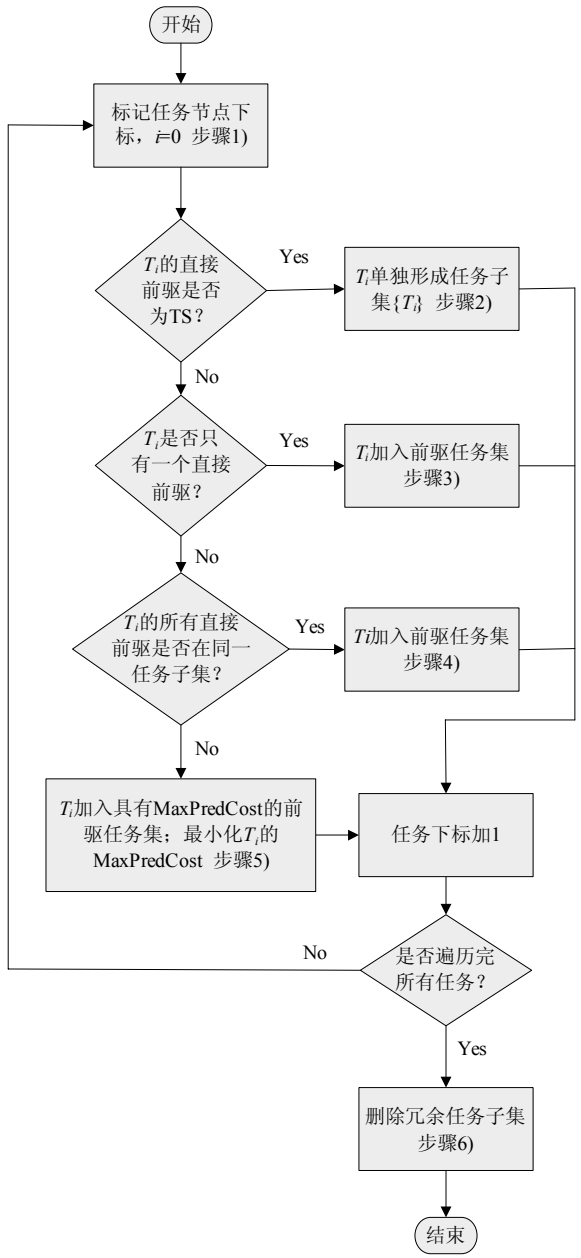

图2 任务子集划分流程图

3) 如果 T_i 的直接前驱不为 TS，且 T_i 只有一个直接前驱 T_j，则直接将 T_i 加入 T_j 所在任务子集 Task_j，形成新的任务集 Task_i，若已经遍历了 T_j 的所有直接后续，删除冗余任务子集 Task_j。

4) 如果 T_i 的直接前驱不为 TS，且 T_i 的多个直接前驱都在同一个任务子集 Task_j 中，则直接将 T_i 加入 Task_j，并将 Task_j 重命名为 Task_i。

5) 如果 T_i 的直接前驱不为 TS，且 T_i 的多个直接前驱分属于 p 个不同的任务子集，此时 $T_i\notin\mathrm{Task}_i$，根据式 (1)，计算 T_i 的最大前驱开销 $\mathrm{MaxPredCost}(T_i)$，并选取具有最大前驱开销的直接前驱 T_j 所在的任务集 Task_j 加入，形成新的任务子集 Task_i，若已经遍历了 T_j 的所有直接后续，删除冗余任务集 Task_j；将 T_i 划分到 Task_i 后，$T_i\in\mathrm{Task}_i$，重新计算最大前驱开销，对于剩余的 p–1 个前驱任务子集，若其中某个前驱任务子集 Task_h 合并到 Task_i 中能减小 T_i 的最大前驱开销，即：$\mathrm{minimize}(\mathrm{MaxPredCost}(T_i))$，则合并 Task_h 到 Task_i 中，否则不合并。

6) 若已遍历完所有任务，则在生成的任务子集中删除冗余集合：若任务子集 Task_i 包含于 Task_j，则删除 Task_i；否则转到步骤 2)。

2.2 负载感知调度

在异构MPSoC环境下，每个处理器核的计算速率和当前负载都是不相同且在随时变化的，负载感知调度综合考虑当前处理器核的负载、处理器核运算速率、任务间通信负载和任务计算开销等多个方面的因素，将划分好的任务子集在运行时环境下动态地映射到合适的处理器核。

Core_i 的负载状态通过 LS_i 表示，分为超载、适载两种状态。核的负载状态需要在每次调度开始之前进行一次更新。

$$\mathrm{LS}_i=\begin{cases}1 & \dfrac{\sum_{k=0}^{|Q_i|-1}r_k}{C_i}\in[0,T_{\mathrm{hreshold}})\\ 0 & \dfrac{\sum_{k=0}^{|Q_i|-1}r_k}{C_i}\in[T_{\mathrm{hreshold}},1]\end{cases}\tag{2}$$

式中，$T_{\mathrm{hreshold}}\in(0,1)$ 表示核负载阈值，可根据应用场景设定；$\mathrm{LS}_i=0$ 表示 Core_i 处于超载状态，即 $\sum_{k=0}^{|Q_i|-1}r_k/C_i\geqslant T_{\mathrm{hreshold}}$，此时不能再向 Core_i 分配任务，而应当选择向适载状态 $\mathrm{LS}_i=1$ 的核分配任务子集。

待分配任务子集到处理器核的映射关系可表示为赋权二部图 $G=(\mathrm{Task},\mathrm{Core},E)$，Task 顶点集对应待分配的任务子集 $\{\mathrm{Task}_0,\mathrm{Task}_1,\cdots,\mathrm{Task}_{m-1}\}$；Core 顶点集对应当前处于适载状态的处理器核的集合 $\{\mathrm{Core}_i,\mathrm{Core}_j,\cdots,\mathrm{Core}_k\}$；$E$ 为 Task 集合中的顶点到

Core集合中的顶点的边，表示任务子集和处理器核的映射关系，边的权重定义为：

$$w_{i,j}=\frac{S_j}{\sum_{k=0}^{|Q_j|-1} r_k+\sum_{k=0}^{|\mathrm{Task}_i|-1} r_k} \tag{3}$$

式中，$\sum_{k=0}^{|Q_j|-1} r_k$ 表示 Core_j 上负载任务的计算开销；$\sum_{k=0}^{|\mathrm{Task}_i|-1} r_k$ 表示 Task_i 中所有任务的计算开销。于是，负载感知调度即在赋权二部图 G 中寻找覆盖顶点集合Task 的最大权匹配。

MPSoC 相较于多处理器系统和分布式系统，其核间通信速率大大提升，处理器核采用计算与传输重叠模式工作，即处理器核能够在运行任务的同时发送/接收数据，因此，在 MPSoC 环境下核间通信已不是影响处理器核执行效率的主要因素。

在赋权二部图 G 的顶点集 $V=\mathrm{Task}\bigcup\mathrm{Core}$ 上确定一个函数 L， L 满足以下条件：

$$L(\mathrm{Task}_i)+L(\mathrm{Core}_j)\geqslant w_{i,j}$$
$$\forall\mathrm{Task}_i\in\mathrm{Task},\forall\mathrm{Core}_j\in\mathrm{Core}$$

则称 L 为图 G 的一个顶点标号。 $L(\mathrm{Task}_i)$ 或 $L(\mathrm{Core}_j)$ 称为顶点 Task_i 或顶点 Core_j 的一个标号。如：

$$\begin{cases}L(\mathrm{Task}_i)=\max\limits_{\mathrm{Core}_j\in\mathrm{Core}}(w_{i,j}) & \mathrm{Task}_i\in\mathrm{Task}\\ \mathrm{L(Core_j)}=0 & \mathrm{Core_j}\in\mathrm{Core}\end{cases}$$

就是图 G 的一个顶点标号。令

$$E_L=\left\{\mathrm{Task}_i\mathrm{Core}_j\mid L(\mathrm{Task}_i)+L(\mathrm{Core}_j)=w_{i,j}\right\}$$

则具有边集 E_L 的 G 的生成子图称为对应于顶点标号 L 的等价子图，记为 G_L 。

基于赋权二部图最大权匹配的负载感知调度如图3所示。

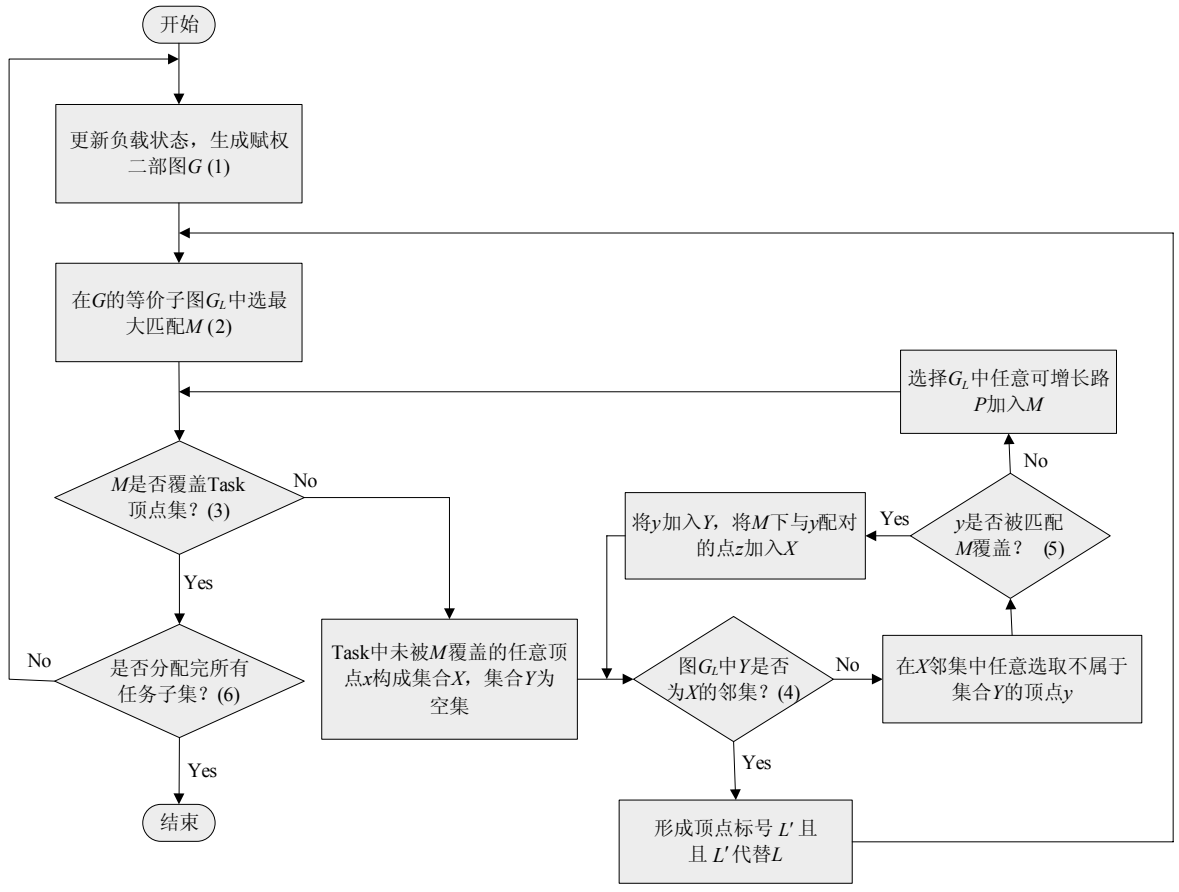

图3　负载感知调度流程图

为了得到一个覆盖 Task 顶点的权最大的匹配，以下分情况讨论：

1) 根据式(2)更新处理器核负载状态，选择处于适载状态的处理器核 $\{\mathrm{Core}_i,\mathrm{Core}_j,\cdots,\mathrm{Core}_k\}$ 作为 Core 顶点集，$|\mathrm{Core}|\leqslant N$；按照任务子集下标递增的顺序，在未处理的 m' 个任务子集选出 $\min\{|\mathrm{Core}|,m'\}$ 个，并将选出的任务子集作为 Task 顶点集 $|\mathrm{Task}|\leqslant m$，根据式(3)计算边权重，生成赋权二部图 G 。

2) 给出 G 的任意顶点标号 L，求出 G_L，在 G_L

中以 Task_i 的下标递增的顺序，选一个最大匹配 M。

3) 若 M 覆盖了 Task 顶点集中的所有顶点，G_L 就是 G 的最大权匹配，转步骤 6)；否则，选择 Task 集合中未被 M 覆盖，且下标最小的顶点 x，令 $X=\{x\}, Y=\varnothing$。

4) 若 G_L 中集合 X 的邻集 $N_{G_L}(X)\supset Y$，则转步骤 5)；否则，$N_{G_L}(X)=Y$ 时，计算

$$\varepsilon=\min_{\substack{\text{Task}_i\in X\\ \text{Core}_j\in Y}}(L(\text{Task}_i)+L(\text{Core}_j)-w_{i,j})$$

且由

$$L'(v)=\begin{cases}L(v)-\varepsilon & v\in X\\ L(v)+\varepsilon & v\in Y\\ L(v) & \text{其他}\end{cases}$$

给出新的顶点标号 L'，用 L' 代替 L。转到步骤 2)。

5) 在 $N_{G_L}(X)\setminus Y$ 中选取一个顶点 y，如果 y 被匹配 M 覆盖，且 $yz\in M$，则 $X=X\cup\{z\}$，$Y=Y\cup\{y\}$，转步骤 4)；否则，在 G_L 中找一条 $M-$可增长路 P，将 P 加入匹配 M，再转步骤 3)。

6) 若未分配完所有任务子集，转步骤1)，否则算法结束。

2.3 算法复杂度分析

LATS算法通过任务子集划分和负载感知调度两个步骤将具有依赖关系的任务调度到处理器核上并行运行。在任务子集划分阶段，算法遍历DAG图中所有任务，对每个任务再遍历其所有直接前驱以计算最大前驱开销用于子集划分，该阶段的复杂度为 $O(n\times d)$，n 为总任务数，d 为DAG图中节点的最大入度。在任务调度阶段，首先需要更新处理器核负载以选择适载处理器核作为调度目标，复杂度为 $O(N)$，N 为处理器个数；其次根据赋权二部图中最大权匹配进行任务子集分配，复杂度为 $O(m\times N)$，m 为任务划分阶段得到的任务子集个数。

考虑在最坏情况下，$d=n-1$，且每个任务单独形成一个任务子集，即 $m=n$，则LATS算法复杂度为 $O(n\times(n-1))+O(N)+O(n\times N)$。在一般情况下，$N\ll n$，因此，LATS整个算法的时间复杂度应为 $O(n^2)$。

3 仿真实验及结果分析

仿真实验利用随机DAG图生成器，设置任务节点数 $m\in\{25,50,75,100,125,150\}$ 和任务的最大出度 $\text{out\_degree}=\{1,3,5,|m|\}$，生成24种不同类型的随机DAG图，随机生成每种类型的DAG图1 000幅，对每幅图用随机函数生成的随机数表示任务计算开销和通信负载，并取1 000次测试结果的算术平均值作为计算结果，参与实验的随机DAG图数量为24M。仿真实验中，T_{hreshold} 取经验值0.85，同时还引入通信计算比CCR(communication to computation Ratio)度量调度算法在不同计算开销和通信负载下的性能。

$$\text{CCR}=\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}c_{i,j}\Big/\sum_{i=0}^{n-1}r_i \tag{4}$$

图 4、图 5 分别在 CCR 为 0.5 和 2 时，LATS、HEFT 和 PLTSF 的平均调度长度的比较图。

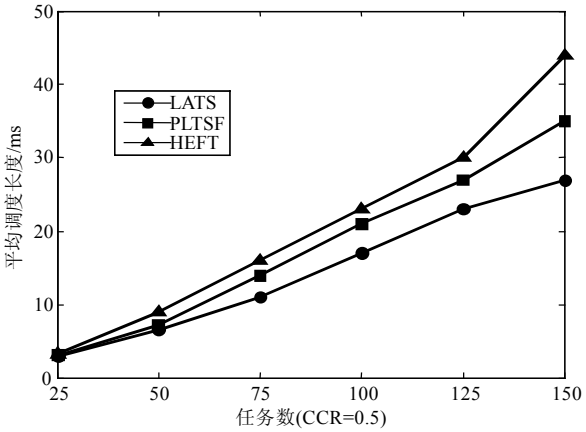

图4 不同任务数的平均调度长度(CCR=0.5)

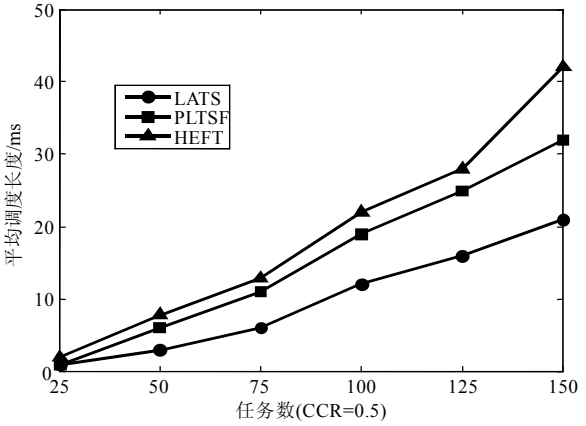

图5 不同任务数的平均调度长度(CCR=2)

从两幅图均可看出，随着任务节点数的增加，系统负载相应增大时，LATS的平均调度长度明显优于HEFT和PLTSF。这是因为LATS考虑了当前系统所有核的负载，将一批任务同时分配到适载处理器核，而HEFT和LATS每次只把一个任务分配到能使其最早执行的处理器核上，只能实现局部最优。

值得注意的是，从图4和图5的对比可以看出，当CCR较大时，LATS算法相同任务数的平均调度长度得到进一步提高。这是因为在LATS算法任务子集划分阶段将具有最大前驱开销的任务划分在了一个任务子集，并在调度阶段将任务子集中的所有任务都调度到处理器核，降低了任务间的通信开销。

图6统计了1 000幅含有150个任务节点的随机DAG图在运行LATS算法、HEFT算法和PLTSF算法的处理器核的利用率。异构MPSoC处理器核利用率是每个异构核利用率的算术平均值。

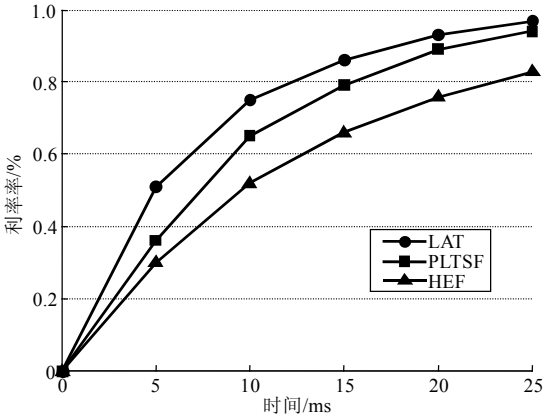

图6　处理器核利用率

从图 6 可知，LATS 算法的处理器核利用率较之 HEFT 算法和 PLTSF 算法有所提升，且由于 LATS 算法每次分配一批任务到适载处理器核，而不是一个一个地分配，其处理器核利用率在任务集运行之初增长较快。

4 结 束 语

任务调度算法使得异构MPSoC的性能得以充分发挥，而处理器核的异构性、运行时负载和任务间依赖关系，是影响异构MPSoC上任务调度算法性能的关键因素。本文提出了一种负载感知的异构MPSoC任务调度算法LATS，在满足任务间依赖关系的前提下，根据计算开销和通信负载将待调度任务集划分为任务子集；在考虑处理器核负载状态的基础上，通过赋权二部图最大权匹配，将任务子集调度到适载处理器核上运行，提高了待调度任务集总执行效率。仿真实验结果表明，LAST算法有效降低了具有依赖关系的DAG任务图的调度长度，提高了处理器核利用率。

参 考 文 献

[1] AGERWALA T, CHATTERIJEE S. Computer architecture: Challenge and opportunities for the next decade[J]. IEEE Micro, 2005, 25(3): 58-69.

[2] POORANI A, ANURADHA B, VIVEKANADHAN C. An effectual elucidation of task scheduling and memory partitioning for MPSoC[C]//2014 IEEE 8th International Conference on Intelligent Systems and Control(ISCO). Coimbatore. India: IEEE, 2014: 295-299.

[3] YAO Xuan-xia, GENG Peng, DU Xiao-jiang. A task scheduling algorithm for multi-core processors[C]//2013 International Conference on Parallel and Distributed Computing, Applications and Technologies(PDCAT). Washington DC, USA: IEEE Computer Society, 2013: 259-264.

[4] LZAKIAN H, ABRAHAM A, SNASEL V. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments[C]//Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization. NJ: IEEE Computer Society, 2009: 8-12.

[5] MITEHELL M. An Introduction to genetic algorithms[M]. Cambrige, MA: The MIT Press, 1996.

[6] QIU Mei-kang, MING Zhong, LI Jia-yin, et al. Phase-change memory optimization for green cloud with genetic algorithm[J]. IEEE Transactions on Computers, 2015, 16(12): 3528-3540.

[7] JIANG Hua, BAO Yun, ZHENG Li-ping, et al. A hybrid algorithm of harmony search and simulated annealing for multiprocessor task scheduling[C]//2012 International Conference on Systems and Informatics(ICSAI). Yantai, China: IEEE, 2012: 718-720.

[8] MISHRA A, TRIPATHI A K. An extention of edge zeroing heuristic for scheduling precedence constrained task graphs on parallel systems using cluster dependent priority scheme[C]//2010 International Conference on Computer and Communication Technology(ICCCT). Allahabad(UP), India: IEEE Communication Society and IEEE UP Section, 2010: 647-651.

[9] DARBHA S, AGRAWAL D P. Optimal scheduling algorithm for distributed memory machines[J]. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(1): 87-91.

[10] ZAID A B, ZENG Hai-bo, MARCO D N, et al. Multitask implementation of synchronous reactive models with earliest deadline first scheduling[C]//2013 8th IEEE International Symposium on Industrial Embedded Systems(SIES). Porto, Portugal: IEEE, 2013: 168-177.

[11] ARABNEJAD H, BARBOSA J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(3): 682-694.

[12] LIU Xin, LIVIU I, RUI Kang, et al. Real-time household load priority scheduling algorithm based on prediction of renewable source availability[J]. IEEE Transactions on Consumer Electronics, 2012, 58(2): 318-326.

[13] TOPCUOGLU H, HARIRI S, WU Min-you. Performance effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274.

[14] 黄殊娟, 朱怡安, 李兵哲, 等. 具有依赖关系的周期任务实时调度方法[J]. 计算机学报, 2015, 38(5): 999-1006. HUANG Shu-juan, ZHU Yi-an, LI Bing-zhe, et al. Real-time scheduling method for dependency period task[J]. Chinese Journal of Computers, 2015, 38(5): 999-1006.

编　辑　蒋　晓