本论文提出一种基于启发式的优化算法-离散型粒子群算法（S-PSO）来解决在多处理平台上任务的调度和映射问题。S-PSO算法在解决组合优化问题时相比其他算法，如：蚁群算法（ACO），具有更加简单的处理过程，同时在求解问题的多样性和计算时间上也具有更好的优势。该算法在求解调度和映射的优化过程是同时进行的，这使得求解空间增大，从而优化结果更具合理性，并且算法在处理大规模问题时仍然能够保持其有效性。

**什么是多处理器片上系统（MPSOC）**

随着商业和大规模计算密集型科研应用对计算平台性能上越来越严格的要求，使得多处理器片上系统(MPSOC)的研究受到越来越广泛的关注。由于多处理器片上系统在单芯上集成了多个异构处理单元，使其可以完成更加复杂的功能，在图像处理、多媒体和嵌入式系统等应用领域前景广阔。多处理器片上系统通过对不同应用场景的定制化设计来满足市场对计算平台性能的苛刻要求，这种灵活的，定制化的设计促使多处理器片上系统成为新一代SOC的主流设计趋势。[1]Tilak Agerwala, Siddhartha Chatterijee. Computer architecture: Challenges and opportunities for the next decade[J]. IEEE Micro. 2005. 25(3): 58-69.

多处理器片上系统(MPSoC)是一种由多个处理器单元和专用功能模块构成的一个复杂的单芯片集成计算系统，通常是为嵌入式应用程序而设计，用于满足应用程序对于系统在计算性能，实时性，功耗与成本等方面的需求[1] W. Wolf, “The future of multiprocessor systems-on-chips,” in Proc. 41st Assoc. Comput. Machinery/IEEE Design Automat. Conf. (DAC), 2004, pp. 681–685.

MPSOC不是仅仅是简单的实现片上多处理器(chip multiprocessor)的片上集成，片上多处理器强调将更多的处理单元放在单片上用于提高单位面积晶体管密度，在设计时并不考虑应用的需求，而MPSOC实现的是定制化的集成芯片，通过定制体系结构来满足不同应用在成本和功耗等方面的需求。

MPSOC体系结构常常用于多任务应用程序的计算，有效的利用MPSOC体系结构的异构性可以有效的满足多任务应用程序在性能，功耗和成本等方面的要求，典型的MPSOC体系结构如下图所示，由3部分组成：

（1） 处理器单元，包括处理器（Processor）、数字信号处理芯片（DSP）、专用集成电路（ASIC）用于执行计算任务；

（2）存储（Memory）模块用于保存数据；

（3）通信网络（Network）用于组件之间的通信。



图1 MPSOC体系结构

**MPSOC上的映射与调度问题**

通信、多媒体和计算机网络等领域，将对计算平台固有的成本压力转换成对多处理器片上系统的性能要求，如何高效的发挥多处理器片上系统的计算性能成为目前研究的热点。

MPSOC上的组件通过片上互连总线连接成为一个整体，为适应不同种类应用程序对不同组件的应用需求MPSOC通常被设计成异构的，AMD公司在2006年提出基于“融合”技术的多核处理器产品路线图，实现了同构片上多处理器系统向异构片上多处理器系统的转化。在MPSOC上我们使用不同种类的处理器来加速应用程序的不同部分，从而优化多任务应用程序的性能。MPSOC结构的复杂性使得对应用性能的优化提出了更高的要求。

近年来，对于MPSOC的研究上围绕系统开发，系统安全，编程模型与任务调度等方面的研究已经拥有了一些重要成果。为了优化应用程序的性能，例如，程序执行时间，需要把一个复杂的任务分解，将分解后的子任务映射到异构处理器上，寻找一组合适的映射关系用于满足实时性，成本与功耗等约束条件成为MPSOC系统亟需解决的关键问题之一。然而多处理器上任务的调度与映射问题属于NP难问题[2]。J. Kim, S. Lee, H.l Shin, Y. Lee and H. Bae, “Effective task mapping and scheduling techniques for heterogeneous multi-core systems based on zone refinement,” Computer Sciences and Convergence Information Technology (ICCIT), 6th International Conference. 2011, pp. 363 – 366.

由于异构嵌入式系统自身结构的复杂性与多任务应用程序规模的不确定性，使得传统方法在处理多处理器上任务的调度与映射问题时失去了有效性，特别是针对大规模的多任务应用程序的调度与映射问题。改进算法的效率并构建任务映射与调度机制成为一个亟待解决的问题。

当前研究提出许多基于启发式的方法来优化任务的映射与调度问题，例如禁忌搜索算法（Tabu Search）[3-4]

T. Wiangtong, P.Y.K. Cheung and W.Luk, “Comparing three heuristic searchmethods for functional partitioning in hardware–software code-sign.” Design Automation for Embedded Systems, 6(4):425–449, 2002.

P. Eles, Z. Peng, K. Kuchcinski and A. Doboli, “System level hardware/software partitioning based on simulated annealing and tabu search.” Design Automation for Embedded Systems, 2:5–32, 1997

遗传算法（Genetic Algorithms）[5]

M. Grajcar, “Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system." In DAC’99: the 36th ACM/IEEE conference on Design Automation, pages 280–285, 1999

以及Kernighan-Lin [6]方法

F. Vahid and T.D. Le, “Extending the Kernighan/Lin Heuristic for Hardware and Software Functional Partitioning.” Design Automation for Embedded Systems,2(2):237–261, March 1997.

以上方法被证明能够在合理的时间内获得对优化问题的一个局部最优的结果。但是这些方法在处理问题时只关注了映射与调度问题的一个方面，即单一的处理调度或映射问题，并且当求解空间的约束条件比较苛刻时，算法在问题求解上的性能就会严重下降，优化的结果往往不尽人意。蚁群算法[7][8][9]

G. Wang, W. Gong, B. DeRenzi and R. Kastner, “Ant colony optimizations for resource and timing constrained operation scheduling.” IEEE Transactions on Computer-Aided Design of Integrated Circuitsand Systems, 26(6):1010–1029, June 2007.

P.C. Chang, I.W. Wu, J.J. Shann and C.P. Chung, “ETAHM: An energy aware task allocation algorithm for heterogeneous multiprocessor.” In Proceedings of DAC ’08, pages 776 –779, june 2008.

F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto and A. Tumeo, “Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems.” IEEE Trans. on Computer-AidedDesign of Integrated Circuits and Systems, 29(6):911–924, June 2010.

是另外一个在处理映射与调度问题中比较常用的优化方法，使用蚁群算法优化问题时对问题的调度与映射的优化是同时进行的，当优化问题的规模比较小时，蚁群算法显示了其有效性和可行性。但是当面对大规模的优化求解问题时，由于优化空间的增加以及计算复杂度的上升，蚁群算法对与该问题的求解上失去了有效性。因此，目前还没有一个通用的，能够高效的处理复杂应用程序在多处理片上系统的调度与映射问题的优化算法。

本文基于抽象模型给出任务的形式化定义，然后进行调度策略与算法的设计

**目标体系结构的抽象（参考论文）**

本文我们抽象出一个通用的异构多处理器片上系统（MPSOC）的体系结构用于进行一般化说明，其形式化定义如下：

A=PUC (1)

其中，P表示一组异构的处理单元，用于处理应用程序的不同部分，C表示一组通信组件，用于进行处理单元之间进行数据的传输。一个抽象异构多处理器片上系统的一个简单示例，如下图所示，它由四个异构的处理单元，分布式内存和单一的系统总线作为通信组件组成的。



图2 抽象目标体系结构

多处理器片上系统的任务调度与映射属于资源受限的优化问题，体系结构中每一个组件都关联一组相关的资源Q（例如：内存大小）[10]

P. Brucker, A. Drexl, R. Mohring, K. Neumann and E. Pesch,“Resource constrained project scheduling: Notation, classification, models, and methods,” Eur. J.Operat.Res.,vol.112, no.1, pp. 3–41, Jan.1999

这些资源可以划分成两类：可再生资源R（renewable）和不可再生资源N（nonrenewable）。

（1）可再生资源R即当组件相关的资源在任务释放以后可以再次提供给后续的任务继续使用。例如：本地内存，当任务结束后内存得到释放提供给后续任务继续使用，但资源的类型也与应用程序使用的方式有关，若是在分配内存时应用程序采用的是静态分配的方式，则当任务结束后内存依然为任务所占有，这时的内存又可以被理解为不可再生资源。

（2）不可再生资源N即当组件相关的资源在分配给单一任务后，即便任务已经执行完成但是其占用的资源也不能再次提供给后续的任务使用。例如：现场可编程门阵列（FPGAs）一般情况下就被认为是不可再生的资源。

在图2中我们假设与处理单元P0，P1和p2相关的资源q0，q1和q2为可再生资源，例如，数据内存单元。与处理单元p3相关的资源q3为不可再生资源，例如，采用静态分配的内存单元。我们用Akq来表示组件pk∈A所关联的资源q∈Q的数量，当任务t分配在组件qk上时任务t所占用的资源数量要满足Akq的要求。

对于图2中目标体系结构上的任务映射与调度问题我们还做如下考虑：

（1）在组件上执行任务时，任务要满足一下两个方面的要求：一、任务的种类和组件可执行任务的种类相符，例如，计算任务只能在计算单元上执行，通信任务只能在数据传输总线上执行；二、任务消耗的资源数量要满足组件当前所拥有的资源数量。

（2）如图2所示，每一个组件都关联一个局部内存，当一个计算任务在组件上运行时，对数据的存取均来至与其相关的局部内存。当任务执行结束以后，其产生的数据也存储在局部内存中，因此，如果两个有数据依赖的任务映射到同一个组件上执行，任务之间的数据传输是不消耗任何资源的，因为并不需要发生实际的数据传输任务[11]。

R. Niemann and P. Marwedel, “An algorithm for hardware/software partitioning using mixed integer linear programming, ” Design Automat. Embedded Syst., vol. 2, no. 2, pp. 125–163, Mar.1997.

此外数据的传输可以通过系统总线或DMA进行传送，但是数据在传送时只能和其中的一个传送组件相关联。通过上面的假设，我们可以很容易的将当前目标体系结构的传输模型向其他模型进行扩展[12]。

T.-Y. Yen and W. Wolf, “Communication synthesis for distributed embedded systems,” in Proc. IEEE/Assoc, Comput. Machinery int. Conf. Comput-Aided Design (ICCAD), 1995, pp. 288-294.

例如在共享内存系统中数据的传输过程将会分为以下两步，一、由源执行单元局部内存到共享内存的数据传输；二、由共享内存到目标执行单元局部内存中的数据传输。在本文中，我们假设：一、所有的数据传输都是直接由局部内存到局部内存进行传输的；二、不同内存单元之间数据传输所消耗的时间只和传输的数据量的大小以及通信总线的带宽有关。

（3）为了简化任务的优化调度，我们假设每个组件在同一时间最多只能运行一个相关任务，即当有多个任务分配在同一执行组件上时，这些任务需要线性的执行。

**多任务应用程序**

什么是多任务应用程序

一个多任务应用程序由N个子任务组成，任务之间具有时序关系和数据的传输依赖，每个子任务在运行时需要占用单一的执行单元。多任务应用程序在执行时具有以下特点：

一、多任务应用程序的并行执行并不意味者所有子任务都需要在同一时间执行，而是在同一时间至少存在一个子任务可以被执行。

二、子任务只有在其所有上游任务已经完成并且获得执行硬件的基础上才能够被执行。

三、多任务应用程序完成的时间为最后一个子任务运行结束的时间。

因此，当多任务应用程序在单CPU系统上执行时，所有的子任务在同一时间上只能有一个在执行，应用程序的执行的总时间和应用程序串行执行的时间相同。为了加快多任务应用程序的执行时间，其执行平台往往是上文所提到了多处理器片上系统，应用程序在执行时，可以并行的执行，但是在调度任务和各个处理单元之间的映射关系上，不同的映射将会导致程序执行时间的不同，例如，在最差的情况下，应用程序中的所有计算任务都被映射到同一个CPU单元上执行，这样应用程序完成的时间和单个CPU执行的时间是相当的，这种映射关系肯定不是我们想要的；我们所期待的结果就是将子任务合理的映射到多个处理单元上，让应用程序充分利用多处理器片上系统的计算优势，加快任务的执行速度，缩短程序的执行时间。

**多任务应用程序的抽象**

在处理异构嵌入式系统上问题的映射和调度时需要将多任务应用程序抽象成能够在目标平台上进行调度的模型。本文我们将多任务应用程序抽象成一个有向无环图（DAG），其形式化定义如下：有向无环图*G* = (*T, E)，*顶点T表示计算任务，E ⊆T×T表示任务之间的依赖关系，即存在e（t’, t）∈E时表明任务t’是任务t的父任务，任务t只有当任务t’执行完成以及和任务t相关的数据传输完成后才能执行。其中，每条边都关联一个数据传输大小的标量，表明由父任务到子任务之间数据传输的大小。一个多任务应用程序的抽象模型如下图所示。拥有这类模型的应用程序主要有科学计算和多媒体应用程序，这类程序在执行时往往具有很的高并行。



图3 多任务应用程序模型

由于上述的多任务应用程序的运行时行为在根据任务进行划分后是可以静态预测的，因此，对多任务应用程序的优化可以是离线进行的。由于进行子任务划分后，任务之间可能会参数较大的数据传输，因此，在进行多任务应用程序的优化时对通信的开销也是必须要考虑的因素。

我们将G=（T,E）和上文定义的目标平台A=PUC进行关联后对多任务应用程序进一步进行以下抽象描述：

（1）任务j(job)被定义为在目标平台的组件上执行的一个活动单元，多任务应用程序的每一个执行任务和任务之间的传输行为都被抽象成一个j，整个多任务应用程序被抽象成一个任务的集合用J来表示。

（2）执行点i被定义成资源和时间的组合，用来代表任务j在组件ak上执行时所需要的最低的资源要求和需要占用的时间，即任务j在ak上执行的时间。在具体的目标平台上，每一个任务j都有一到多个执行点可供选择来执行自身的任务，只要执行点满足任务j所需的约束执行的条件，若是不满足任务j是不能分配到该执行点上执行的，例如，计算任务j不能被分配到传输总线上进行来执行。此外，并不是所有的处理单元都可以用来执行计算任务，例如。一个任务包含了不能合成的结构。任务j在不同的执行点i上的组合，构成了应用程序在硬件上的解决方案。执行点的集合I包含了任务集合J在目标平台A上所有可用的执行点，表格I对多任务应用程序G在目标平台A上执行点的集合I进行了展示。表格中的每一项表明了任务j在执行点i上执行时需要的时间（例如，时钟周期）和需要消耗的资源数量（内存大小），对于每一个计算型的任务j，在P3这个计算单元上可以有两个不同的执行点，这是因为这两个执行点是根据任务在P3上消耗的资源数量不同而产生的，是任务在执行时占用资源和时间的平衡的结果，一般情况下，占用的资源数量越多，任务运行时消耗的时间就越小。如表1中执行点i3和执行点i4同样是对物理组件Pe3的映射，但是却是对资源数量和执行时间不同的平衡。

表1 任务J和执行点I之间的映射关系表

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| job | pe0 | | pe1 | | pe2 | | pe3 | | | | cc | |
| i0 | | i1 | | i2 | | i3 | | i4 | | i5 | |
| t | rr | t | rr | t | rr | t | rr | t | rr | t | rr |
| A | 3 | 6 | 7 | 4 | 3 | 8 | 6 | 3 | 2 | 9 | null | null |
| B | 5 | 6 | 2 | 2 | 7 | 9 | 2 | 3 | 1 | 12 | null | null |
| C | 6 | 8 | 2 | 9 | 2 | 10 | 10 | 9 | 12 | 2 | null | null |
| D | 9 | 12 | 4 | 8 | 19 | 5 | 7 | 13 | 7 | 9 | null | null |
| E | 3 | 6 | 7 | 4 | 3 | 8 | 6 | 3 | 2 | 9 | null | null |
| F | 8 | 2 | 7 | 12 | 1 | 12 | 2 | 9 | 11 | 7 | null | null |
| G | 12 | 3 | 12 | 7 | 1 | 9 | 11 | 12 | 9 | 10 | null | null |
| (A,B) | null | null | null | null | null | null | null | null | null | null | 2 | 8 |
| (A,C) | null | null | null | null | null | null | null | null | null | null | 4 | 8 |
| (B,E) | null | null | null | null | null | null | null | null | null | null | 5 | 8 |
| (C,E) | null | null | null | null | null | null | null | null | null | null | 7 | 8 |
| (C,F) | null | null | null | null | null | null | null | null | null | null | 3 | 8 |
| (D,F) | null | null | null | null | null | null | null | null | null | null | 1 | 8 |
| (E,G) | null | null | null | null | null | null | null | null | null | null | 2 | 8 |
| (F,G) | null | null | null | null | null | null | null | null | null | null | 2 | 8 |

（3）函数γ：是执行点集合I到目标体系结构的A的映射关系，即I → A；其返回值为执行点i所关联的目标体系结构上的物理组件pk，其中，物理组件Pk所关联的资源数量Akq 也是执行点i可用的资源的数量。根据上文关于目标平台A的构建模型可以看出，对执行点i的使用是可以分时共享的。如表I所示，执行点i2映射到的物理组件为pe2。

（4）函数δ：是任务j和执行点i的组合到执行时间t的一个映射关系，即当任务j分配到执行点i上时任务j运行时所需要的执行时间，形式化表达为：J×I→T。如表1所示，任务A在执行点i2上执行时所需的时间t为7。

（5）函数σ：是任务j和执行点i的组合到占用资源数q∈Q的一个映射关系，即当任务j分配执行点i上时任务j运行时需要占用的资源数量q，其形式化定义为：J×I→Q，并且任务j在执行点i上可以运行所必须满足的条件为σ（j,i）≦Akq，其中ak=γ（i）。如表1所示，任务A在执行点i2上执行所需的资源数量为4。

本文中对应用程序的划分是离线进行的，所以在上述映射关系中，对于每一个任务j在执行点i上关于消耗资源数量q和执行时间t的获取上，采用的方法为通过模拟评估的方法或者通过对应用程序代码和数据进行时间和空间复杂度的分析后获得的。同时，为了简化对问题的求解，突出论文中提出的优化方法与思路并将应用问题抽象化表达，在后续的模拟实验上，我们假定上述函数的值都是已知的，其具体的表达形式如图1所示。

**多任务应用程序在目标平台上的优化目标**

本文中我们的主要优化目标是多任务应用程序在目标平台上的执行时间，即由应用程序开始执行到应用程序执行结束的一个时间段。我们假定每一个任务j开始执行的时间为S(j)，任务j在满足约束的情况下被分配到执行点i上执行，则任务j执行结束的时间为：

Hj = S(j) + δ(I,j) (2)

其中，应用程序第一个开始执行的任务的开始时间采用相对时间计算，默认起始时间为0,.

由此可知，应用程序总的执行时间ET为：

ET = max（Hj） ∀j ∈ J (3)

有上述公式可知，应用程序的执行时间和每个子任务j的开始时间和任务子j在执行点i上的执行时间有关。因此为了最小化应用程序的执行时间，我们需要在最小化子任务的执行时间，但是由于硬件平台相关组件的分时共享，目标平台资源的约束以及任务之间的依赖关系等特性，在进行任务优化的同时还需考虑以上约束条件。

（1）目标平台上将资源Q划分为可更新资源R和不可更新资源N，因此（1）资源是可更新资源时要求，任务j所需要的资源数量小于等于执行点i所拥有的数量；（2）资源是不可更新资源时，与执行点i相关的资源将会被任务j消耗，这要求在调度时分配在执行点i上的所有任务消耗的资源总和小于等于执行点i关联的资源数量。在形式化资源约束的表达式时根据资源类型分为以下两个公式来表达：

σ（j,i）≦Akq 其中，∀j∈J, q∈R (4)

(5)

（2）关于任务之间的依赖约束要求任务j的开始执行点S(j)必须等到上游父任务已经完成，并且其分配的执行点i所关联的组件γ(i)处于空闲状态。其形式化定义如下：

max[H(j’), avail(γ(i))] ≦ S(j) , ∀j’∈prec(j) (6)

其中prec(j)是指任务j的父节点集合，avail(γ(i))返回执行点i关联的组件可用时间点。

根据公式(2)我们可以得到公式(6)的一个推导公式如下：

max[S(j’) +δ(I’,j’), avail(γ(i) ] ≦ S(j) , ∀j’∈prec(j) (7)

由公式(7)可以看出，在目标平台上的多任务应用程序的调度与优化问题上存在很强的依赖性，任务j的执行时间不仅和其父任务有关，还与其对应的执行点i上的其他任务有关。例如，有两个任务j’和j，这两个任务在应用程序中的逻辑上下文中并没有时序和数据上的依赖关系。（1）当j’被分配到组件I’上执行时，如果任务j也被分配到相同的执行点I’上，（或者不同的执行点i，但是γ(I’) =γ(i)）时，任务j’和任务j将会分时的在执行点I’上串行的运行。（2）当j’被分配到组件I’上执行时，如果任务j被分配到不同的执行点i并且γ(I’) ≠γ(i)时，任务j’和任务j才可以被并行的执行。由上面的例子可以看出，对任务j在执行点i上的映射对应用程序的优化结果有着重要的影响，在进行任务的调度时并不是单独的考虑任务在执行点的执行时间的大小，而是需要综合考虑任务的映射和调度问题。映射是为了最大化任务的并行执行，调度是为了寻找任务最佳的执行点，因此，在求解问题时必须同时考虑调度和映射问题，这样对问题的求解才会更加合理。

**相关研究进展**

近年来，MPSOC相关研究引起了广泛的关注，围绕着编程模型、任务调度、系统开发与系统安全等方面的研究成果层出不穷，其中用于把一个复杂任务分解，将其映射到异构多处理器上，在满足约束的情况下，实现任务性能要求是MPSOC系统研究的关键问题之一。任务调度算法主要针对抽象任务模型进行可调度性分析和调度算法设计，在提高系统的可调度性的同时又保证算法的效率是问题的关键[13]

Baruah S, Nathan F, “The feasibility analysis of multiprocessor real-time systems” Proc of the EuroMicro Conf on Real-Time System. Los Alamitos, CA: IEEE Computer Society Press, 2006: 85-94.

针对任务的调度与映射问题的算法，可以根据任务类型被分为两类：在线优化算法和离线优化算法，本文我们讨论的重点主要集中在后者身上。

任务调度的相关研究最早是由Liu和Layland[14]提出的

Liu C, Layland J, “Scheduling algorithms for multiprogramming in a hard real-time environment”.

Journal of the ACM, 1973, 20(1): 4-61.

周期任务模型，这类模型是后来很多实时任务模型的基础，其主要优势在于忽略了实现细节，对研究对象进行了高层次抽象。在处理多处理器任务划分问题时，上述模型在求解问题时是基于一些重要假设，即任务的独立性，任务的划分执行和无通信开销。然而在多处理器片上系统中运行的多任务应用程序时，任务之间的数据依赖，时序关系，和通信开销使得周期任务模型并不能很好的描述实际系统的需求，研究者不断放松对调度问题的限制条件进行问题模型构建期望达到模型与实际相吻合的情形。

当前阶段在处理嵌入式系统上多任务的调度问题时，通常采用DAG模型，用一个有向无环图DAG(T,E)来对问题进行建模，其中，V是节点集合，一般用于表示计算任务，E是边集合，用于表示任务之间的时序关系，每条边还可带有一个标量用于表示数据传输量的大小，基于DAG模型的问题在将将任务分配到目标平台上执行，在满足约束的情况下，期望达到预期的性能要求。有向无环图模型和周期任务模型相比更加的实用，在表达任务之间的依赖关系和通信开销上有不可比拟的优势，因此，在进行多处理器片上任务的调度研究中被广泛使用。

在基于DAG建模的基础上，近年来研究者有引入了新的计算方法，Niemann与Marwedel [11]提出的整数线性规划法（ILP）在基于DAG的基础上进行任务的优化，在对问题进行形式化表达时增加了对目标平台的异构性和数据通信开销的因素。然而在针对某一个特定硬件上任务的执行是单一的，并没有考虑到同一硬件因为对资源消耗书和运行时间的平衡而产生的不同的执行点的抽象。相对而言，我们在进行目标平台的表达上更加符合实际应用，首先，是对资源类型的划分上，包括了可更新和不可更新资源；其次，在资源消耗和时间上的平衡因素的衡量，产生的对单一硬件上的不同执行点的抽象。带有资源约束的任务映射与调度问题是NP难问题，常规的方法在解决该类问题时效果往往不尽人意，这时基于启发式的优化算法的产生，对这类问题的优化提供了一个新的思路。

例如,基于DAG模型上多处理器任务调度问题在文献[12]

Annie S Wu, Han Yu, Shiyuan Jin, “An inreamental genetic algorithm approach to multiprocessor scheduling”. IEEE Trans on Parallel and Distributed System, 2004, 15(9): 824-834.

提出了基于遗传算法进行多处理器任务调度的优化算法，该算法在优化问题上具有灵活，自由的特性，和传统的调度算法相比较其调度性能更好，缺点是运行时间复杂度较高。

基于启发式的优化算法在求解问题时，能够保证在满足约束的前提下获得局部最优的结果。但是在对问题建模时往往只专注问题的一个方面进行优化处理，即隔离了问题求解时的调度和映射问题，只从单一的角度出发来求解问题的可行解。例如在已知任务和硬件资源的映射前提下，优化每个子任务的最优开始时间。[17]

D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik and P. Wong, “Theory and practice in parallel job scheduling,” in Proc. Job Scheduling Strategies Parallel Process. (IPPS), 1997, pp. 1-34.

这些算法只关注多任务应用程序的调度问题，在进行建模时，优化空间的合理性从根本上限制了优化结果的质量，例如在文章[13]

S. J. Beaty, “Genetic algorithms versus tabu search for instruction scheduling,” in Proc. Int. Conf. Neural Netw. Genetic Algorithms, Feb. 1993, pp. 496–501.

中采用的模拟退火算法与禁忌搜索算法以及[14]

M. Grajcar, “Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system,” in Proc. 36th Assoc. Comput. Machinery/IEEE Conf. Design Automat. (DAC), 1999, pp. 280–285.

中的遗传算法。

本文中我们同样采用启发式的算法来优化问题的结果，但是在对问题求解时，对问题的建模方法允许同时考虑映射和调度问题，增大了问题的求解空间，虽然求解空间的增加会带来求解复杂度的提升，但是带来的效果，是使得优化结果更加合理，因为对问题的优化是离线进行的，所以时间的增加，并不是我们所关注的重点，我们更加关注对求解质量的比较。

论文[3]

J. I. Hidalgo and J. Lanchares, “Functional partitioning for hardware software code sign using genetic algorithms,” in Proc. 23 rd EUROMICRO Conf. 1997, pp. 631-638.

中的GA算法，论文[4]

T. Wiangtong, P.Y.K. Cheung and W.Luk, “Comparing three heuristic searchmethods for functional partitioning in hardware–software code-sign.” Design Automation for Embedded Systems, 6(4):425–449, 2002.

和[5]

P. Eles, Z. Peng, K. Kuchcinski and A. Doboli, “System level hardware/software partitioning based on simulated annealing and tabu search.” Design Automation for Embedded Systems, 2:5–32, 1997.

中的TS与SA算法证明在求解多任务应用程序的调度问题时都具有比较好的性能，

但是这些方法在优化问题时都存在着问题，这些方法，在问题建模时对子任务的依赖关系以及数据传输没有进行约束，问题的优化调度基本上是在不受约束的环境下执行的，这可能导致产生的结果是不可行解。为了解决这个问题人们又提出了带有回溯机制的优化方案[12], J. Teich, T. Blickle, and L. Thiele, “An evolutionary approach to system level synthesis,” in Proc. 5th Int. Workshop Hardware/Software CoDesign(CODES), 1997, pp. 167-171.

但这种机制不但增加了问题的求解时间，同时也降低了问题的优化空间，算法容易陷入局部最优。对于带有任务间通信的优化模型，近年来也产生许多重要的成果[4]

T. Wiangtong, P.Y.K. Cheung and W.Luk, “Comparing three heuristic searchmethods for functional partitioning in hardware–software code-sign.” Design Automation for Embedded Systems, 6(4):425–449, 2002.

[28]M. Purnaprajna, M. Reformat, and W. Pedrycz/ “Genntic algorithm for hardware-software partitoning and optimal resourse allocation,” J. Syst. Architect, vol, 53, no. 7, pp. 339-354, 2007.

上述论文在进行建模时虽然考虑了通信开销，但是对物理平台的抽象上并没有考虑不同的通信总线以及对总线资源的分时共享特性。在我们的模型中，充分的考虑了通信开销和系统总线这一基本事实，分别将通信任务和系统总线进行一般化抽象，以方便其在DAG模型中进行调度和优化。近几年比较流行的ACO[7979808080]

F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto and A. Tumeo, “Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems.” IEEE Trans. on Computer-AidedDesign of Integrated Circuits and Systems, 29(6):911–924, June 2010.

算法,也被证明在进行多任务应用程序的优化上具有很好的优化效果，但是当求解问题规模增加，问题的优化空间变大时，ACO算法的优化效率就会显著降低，这种性能上的降低具体表现在两个方面，一、优化结果的质量显著下降，算法在约束条件变多，求解空间增加时很容易陷入局部最优；二、算法在进行爹带运算时，其收敛速度变慢，达到相同的效果，蚁群算法需要更多的迭代次数。本文中我们在对求解问题进行建模时不但充分考虑了问题的各个方面，并且能过使得算法在充分满足软硬件约束条件的前提下，高效的实现多任务应用程序在目标平台上的优化任务。本文后续章节，我们将通过实验在相同的模型下比较ACO算法和离散粒子群算法对同一模型的优化效果，来验证离散型粒子群算法对该模型的适用性和有效性。

**什么是PSO算法**

随着计算机学科的发展，科学家将自然界中生物的简单智能行为引入计算机学科当中，用于帮助求解复杂的优化计算问题，例如，上文中提到的模拟退火算法、禁忌搜索算法、蚁群算法、遗传算法以及粒子群优化算法等。上述众多的算法中，粒子群优化算法是通过模拟飞鸟的迁徙行文而产生的一种优化方法。它的思想源自于自然界动物群体的智能行为。在对动物群体智能的研究中，研究人员发现自然界的动物在群体行为中呈现出高度的组织性和规律性。由于受到群体智能的启发J. Kennedy 和 R. C. Eberhart在1995年提出了粒子群优化算法[5]

J. Kennedy and R. C. Eberhart,“Particle swarm optimization,” in Proc. IEEE Int. Conf. Neural Netw., 1995, pp. 1942–1948.

粒子群算法的基本思想是：每个粒子（飞鸟）在群体迁徙的空间中移动时需要根据两种重要的信息来不断更新飞行的速度和方向，这两种信息分别是：一、粒子自身的经验（个体认知）；二、整个迁徙群体的总体经验（外界影响）。

根据这一基本思想，对群体行为进行形式化后产生的粒子群优化算法基本流程如下：

首先，将群体空间抽象描述为：假设群体中粒子的规模为N（飞鸟的个数），群体在一个n维空间中搜寻目标T（三维空间的觅食行为），群体中的每个粒子i都具有两个基本属性xi和vi，其中xi是一个n维空间向量，代表粒子的位置；vi是一个n维空间向量，代表粒子的速度。其次，每个粒子附带一个历史最优位置向量用于表达粒子的个人认知，以及一个全局最优粒子用于表达群体对每个粒子的外界影响，为了更新全局和局部最优粒子，算法还需同一个适应值函数，用于评估当前位置的好坏，例如，可以通过求解粒子和目标之间T之间的欧式距离来进行评估。根据以上设定粒子群的目标寻找过程将包括以下步骤：

（1） 对空间中的每个粒子i（i=1,2，…，N），初始化粒子i的位置xi和速度vi，并将该粒子的历史最优位置pbesti初始化为当前位置，然后，在粒子群N中选取和目标T最近的粒子来初始化全局最优粒子gbest。

（2）根据全局最优位置gbest和每个粒子的历史最优位置pbesti来更新粒子i的飞行速度，更新公式为：

（6）

（3）根据粒子i的飞行速度vi更新粒子当前的位置，更新公式为：

（7）

（4）计算适应值函数来确定是否需要更新粒子的历史最优位置pbesti以及全局最优位置gbest。

（5）如果算法满足迭代条件则结束并返回全局最优位置gbest，否则，返回步骤（2）继续计算。

下图给出了算法的伪代码。根据以上表述可以看出粒子群算法的基本流程十分的简单，去其他算法相比，粒子群算法在编程实现和计算效率都有很大的提高。但是上述的粒子群优化算法也存在一下不足：首先，算法容易陷入局部最优，在优化过程中会出现过快收敛的情况；其次，上述的粒子群优化算法只适用于连续粒子空间，对于离散型的问题将失去有效性。

|  |
| --- |
| **Procedure PSO** |
| Initialization;  While terminal condition not meet  For each particle i (i=1,2,…,N)  Velocity updating; (5)  Position updating; (6)  End for  For each particle i (i=1,2,…,M)  Update the ***pbestk*** and ***gbest***.  End while |
| **End procedure** |

图4 粒子群算法伪代码

由于多处理器片上任务的调度与映射问题是离散型问题，所以传统的离散粒子群优化算法并不适应于求解该问题，因此，我们在方法的选取上采用一种新型的集合型离散粒子群优化算法

W.N. Chen, J. Zhang, Chung, H.S.H., W.L. Zhong, W.G. Wu and Y.H. Shi, “A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems,” IEEE Trans. Evol. Comput. Vol. 14, no. 2, pp. 278-300, April 2010.

来求解该问题。该算法通过对连续空间粒子群算法重新进行集合化定义成功实现了粒子群算法在离散问题上的应用。

集合型离散粒子群算法具有一下特性：首先，采用基于集合的方法重新定义对问题的解（即粒子的位置）；其次，在集合空间上重定义粒子群的速度和位置更新操作。最后，通过对粒子和更新操作的集合化定义后，集合型离散粒子群算法在保持传统粒子群优化算法的简答结构和搜索方式的同时也适应于对离散性问题进行求解。