文章编号: 1001-0920(2009)09-1385-05

基于自适应蚁群算法的软硬件划分

张煜东[a], 吴乐南[a], 韦　耿[a], 吴含前[b], 郭永亮[c]
(东南大学 a. 信息科学与工程学院, b. 软件学院, c. 移动通信国家重点实验室, 南京 210096)

摘　要: 为了更好地解决软硬件双路划分问题,提出一种自适应蚁群算法. 基本思想是:对状态转移概率与信息素挥发因子,采取自适应调节策略. 保证了算法前期蚁群的随机性较大,可以充分全局搜索;算法后期蚁群的随机性降低,以使算法在较短的时间内收敛. 对不同节点的控制数据流图进行仿真实验,表明在同等条件下,相对于改进模拟退火、改进禁忌搜索、改进蚁群算法以及 DCG3A 方法,所提出算法的命中率与收敛时间结果均更优. 节点规模越大,优势尤其明显.

关键词: 嵌入式系统; 协同设计; 软硬件划分; 蚁群算法
中图分类号: TN911.73　　**文献标识码**: A

Hardware/software partition using adaptive ant colony algorithm

ZHANG Yu-dong[a], *WU Le-nan*[a], *WEI Geng*[a], *WU Han-qian*[b], *GUO Yong-liang*[c]
(a. School of Information Science and Engineering, b. School of Software, c. National Mobile Communication Research Laboratory, Southeast University, Nanjing 210096, China. Correspondent: ZHANG Yu-dong, E-mail: zhangyudongnuaa@gmail.com)

Abstract: In order to solve the hardware/software bi-partitioning problem more efficiently, a novel adaptive ant colony algorithm (AACA) is proposed. The basic idea is to adaptively adjust the state transform probability and the pheromone evaporation factor, which ensures that the randomness of the ant colonies is high enough at the initial for global search and low at the later stage for local search for faster convergence. Experiments synthesize different nodes control data flow graphs, and show that the proposed method has superiority over improved simulated annealing (ISA), improved tabu search (ITS), improved ant colony algorithm (IACA), and DCG3A in terms of global convergence rate and computation time. The more the nodes are, the obvious the superiority is.

Key words: Embedded system; Co-design; Hardware/software partitioning; Ant colony algorithm

1 引　　言

传统的嵌入式系统设计方法将硬件和软件划分为两个独立的部分,由硬件工程师和软件工程师根据设计经验和一些辅助工具分别设计硬件和软件,最后集成验证. 如果发现错误,不得不推翻重来,这样便增加了设计次数,既费时又费成本.

近年来,一种新的软硬件协同设计方法逐渐被学者所关注. 该方法强调硬件工程师与软件工程师的合作,并行设计系统的软硬件,并在设计的全过程中控制软硬件的一致性和系统的正确性. 由于采用了统一的描述方法,协同设计可以在早期对系统设计的正确性进行验证,从而及时发现设计错误.

由于协同设计是一个 NP-C 问题,为了能够对其快速求解,学者们进行了大量的研究. Ernst 等[2]提出爬山法, Saha 等[2]提出用遗传算法(GA), Gajski 等[3]提出一种改进的模拟退火算法(ISA), Vahid 等[4]提出一种改进禁忌搜索(ITS), Wu 等[5]提出一种改进的蚁群算法(IACA),熊志辉等[6]提出采用遗传算法与蚂蚁算法的动态融合(DCG3A), Kalavade[7]提出用 GCLP/LBS 算法. 这些算法有一个共同点:通过定义某种启发信息,指导搜索过程逐步向最优解收敛.

上述算法虽能较成功地解决了 NP-C 问题,但本身也存在缺陷. 爬山法采用"以退为进"的策略寻

收稿日期: 2008-10-17; **修回日期**: 2009-01-15.
基金项目: 国家自然科学基金项目(60473065,60572063,60872075,60802006); 高等学校科技创新工程重大项目培育基金项目(706028); 国家 863 计划项目(2008AA01Z227).
作者简介: 张煜东(1985—),男,江苏苏州人,博士生,从事蚁群算法、神经网络等研究; 吴乐南(1952—),男,安徽枞阳人,教授,博士生导师,从事多媒体信号处理等研究.

优,但易陷入局部最优; GA 执行到一定阶段后,向最优解收敛速度过慢;ISA 模拟物质材料的冷却与结晶过程,通过退火温度控制搜索过程,但问题规模较大时,搜索进入热平衡过程的时间较长;ITS 模拟人类智力过程,但数据存取操作频繁;DCG3A 无法判断由 GA 转入 ACA 的合适时刻;GCL P/LBS 对目标系统结构的依赖性较大.

实验表明,当系统的节点数较少时,传统的启发式算法能够在较短时间内找到全局最优点.但是,当系统的节点数较多时,这些算法不仅耗时指数增长,而且极易收敛到局部最优,无法找到全局最优点.

综上所述,传统的方法面临两个问题:1) 极易收敛到局部最优;2) 消耗的时间随着节点数的增加而指数增长.为了解决上述两个问题,本文提出一种自适应蚁群算法(AACA),用来求解软硬件划分问题.其基本思想是:采用 Ant-C 模型,对状态转移概率与信息素挥发因子,采取自适应调节策略.在算法初期将蚁群搜索的随机性能提高,同时赋予信息素挥发因子一个较大值,以利于蚁群的全局搜索;在算法后期将蚁群的随机性能降低,同时逐渐减小信息素挥发因子,以利于蚁群的局部搜索.

2 软硬件划分介绍

传统芯片是软硬件分开的,由于工艺水平的提高,现在的芯片上既可以集成硬件(CPLD,FPGA,ASIC),还能集成软件(CPU,DSP),图 1 是一个可重构片上系统(SoC).

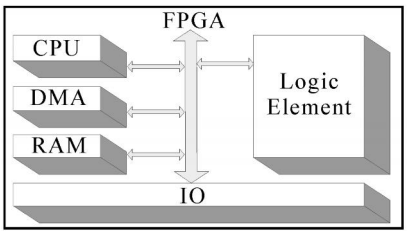


图 1 可重构 SoC

对于给定的应用系统,先将它分为一定粒度大小的任务,然后用控制数据流图(CDFG)来描述.CDFG是一个有向无环图(DAG),如图 2 所示.每个节点代表 1 个任务,每条边代表节点间通信过程,此过程也需要一定的系统开销.但由于节点任务执行时间的开销远大于节点间的通信时间开销,一般忽略不计.

若系统的任务节点数为 N,则第 $i(i = 1,2,\cdots,N)$ 个节点采用第 $j(j = 0,1,0$ 表示软件,1 表示硬件) 种开发方式,对应有 4 个属性:1) 执行时间 $t(i,j)$;2) 面积 $S(i,j)$;3) 功耗 $P(i,j)$;4) 开发代价 $C(i,j)$.若系统的最大硬件面积为 S,最大功耗为

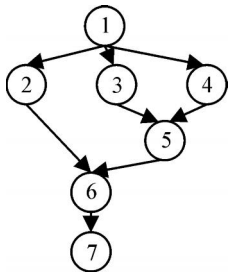


图 2 7 节点系统的 CDFG

P,最大开发代价为 C,设划分方式 x 用一串二进制序列表示,如{0100110},则软硬件划分可以等效为如下极小化问题:

$$x = \arg\min f(s) = \arg\min \sum_{i=1}^{N} t[i,x(i)]; \quad (1)$$

$$\text{s.t.} \sum_{i=1}^{N} S[i,x(i)] \le S_H, \quad (2)$$

$$\sum_{i=1}^{N} P[i,x(i)] \le P, \quad (3)$$

$$\sum_{i=1}^{N} C[i,x(i)] \le C, \quad (4)$$

其中 $x(i)$ 表示对第 i 个节点采用的开发方法.

3 基本蚁群算法简介

设 $b_i(t)$ 表示 t 时刻位于城市 i 的蚂蚁数,n 为城市规模,m 为蚂蚁总数,C 为城市集合,R 为路径集合;$\tau_{ij}(t)$ 为 t 时刻路径 (i,j) 上的信息素,ρ 表示信息素挥发系数,$1-\rho$ 表示信息素残留因子;d_{ij} 为城市 i 与 j 的距离,$\eta_{ij}(t)$ 为 t 时刻路径 (i,j) 上的启发函数,一般取 $\eta_{ij}(t) = 1/d_{ij}$;Tabu$[k]$ 表示蚂蚁 k 走过的城市,Allow$[k]$ 表示蚂蚁 k 未来允许走的城市,满足 Tabu$[k]$ + Allow$[k]$ = C.则 ACA 算法可以用如下伪码叙述.

3.1 算法步骤

Step1: 初始化.

将 m 只蚂蚁随机放置在 n 个城市中,并令所有路径上信息量相等,即 $\tau_{ij}(0) = \text{Const}$.

Step2: 迭代过程.

While 不满足终止条件 do

For i = 1 to n do// 城市循环

For k = 1 to m do// 蚂蚁循环.

对于蚂蚁 k,如果 t 时刻在城市 i,则转移到城市 j 的概率为

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}}{\sum_{s \in \text{Tabu}[k]} [\tau_{is}(t)]^{\alpha}[\eta_{is}(t)]^{\beta}}, & j \in \text{Allow}[k]; \\ 0, \text{ otherwise.} \end{cases} \quad (5)$$

其中:α 表示信息启发式因子,表示遗留在路径上的

信息量的重要性;β表示能见度因子,表示能见度的相对重要性. 对残留信息量按照下式进行更新:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t), \quad (6)$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t). \quad (7)$$

其中:$\Delta\tau_{ij}(t)$ 表示本次循环中路径(i,j)上的信息素增量,$\Delta\tau_{ij}^{k}(t)$ 表示第 k 只蚂蚁在本次循环中在路径(i,j)上的信息素增量. $\Delta\tau_{ij}^{k}(t)$ 有 3 种不同求法,对应了 Ant-C 模型,Ant-Q 模型与 Ant-D 模型. 由于 Ant-C 模型利用整体信息,求解优化问题性能较好,本文采用该模型,为

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q/L_k, & \text{第 } k \text{ 只蚂蚁经过}(i,j); \\ 0, & \text{otherwise}. \end{cases} \quad (8)$$

其中 L_k 表示第 k 只蚂蚁在本次循环中所走路径的总长度.

End

End

3.2　算法的正负反馈机制

由于蚁群具备系统的 3 个特征,即多元性、相关性和整体性,可将蚁群看作一个系统. 而系统的自我更新与完善,需要正反馈与负反馈的结合.

正反馈就是信息素 τ 的堆积. 根据式(8),蚂蚁在随机走完一次循环后,对于发现的较短路径,会遗留下较多的信息素. 然后通过式(5),信息素越多,就会更大概率地吸引蚂蚁在下一次循环中选择该路径. 这样不停循环,最终,所有蚂蚁都会选择该路径. 然而,仅有正反馈会让蚁群易陷入局部最优,负反馈可以保证跳出局部最优点. 负反馈即是随机搜索技术与信息素挥发技术. 根据式(5),蚂蚁构造解的过程是按照概率进行的,即使某条路径的选择概率很大,蚂蚁还是可能选择其他路径. 再根据式(6),每次循环之后,路径上的信息素都要强制挥发,以保证信息素不会太快地集中在某条路径上. 因此,负反馈技术的引入,部分抵消了正反馈的效果,使得蚁群的正负反馈结合,平衡了收敛时间与全局收敛概率这对矛盾.

3.3　算法的不足

ACA 对大规模问题处理效果欠佳,根源在于正负反馈不处于一个"和谐"状态. 代表正反馈与负反馈的参数,在算法运行期间是固定不变的,这与现实情况相违背. 理想的蚁群算法应该具有以下 2 个特性:

1) 在算法前期,负反馈较强,利于全局搜索;在算法后期,正反馈较强,利于局部搜索.

2) 参数应随算法的运行而自适应变化,而不应该采用恒值或突变的形式. 因为参数对应着算法的正负反馈效果,而系统本身的正负反馈平衡点是时刻变化的,所以参数也应该自适应变化.

4　自适应蚁群算法

根据3.3节对 ACA 的分析,提出一种自适应蚁群算法(AACA),思想如下: 算法前期,使系统的负反馈增强,即增加蚁群的随机性,使其可以充分进行全局搜索. 经过一定代数后,蚁群已接近最优解,再使系统的正反馈增加,即减少蚁群的随机性,使之充分地局部搜索到最优解.

4.1　信息素挥发因子的改进

信息素挥发因子 ρ 变化如下:

$$\rho(t) = \max[\gamma\rho(t-1), \rho_{\min}]. \quad (9)$$

其中:$\rho_{\min}$ 为 ρ 的最小值,防止 ρ 过小影响算法性能;γ 是预设的衰减常数. 式(9)迫使信息素挥发因子 ρ 从最大值逐渐降低,但又不至于太低而影响算法性能.

对比式(6)可以发现,传统的 ρ 是一个恒定值. 而改进之后,在算法初期 ρ 取大值,提升算法随机性,利于全局搜索;在算法后期 ρ 取小值,降低算法随机性,利于局部搜索. 显然比传统的取 ρ 恒值性能好.

4.2　转移概率的改进

第 k 个蚂蚁从城市 i 转到城市 j 的概率按照下式变化:

$$P_{ij}^{k} = \begin{cases} \dfrac{[\tau(i,u)]^{\alpha}[\eta(i,u)]^{\beta}}{\sum_{u\in \mathrm{Allow}[k]}[\tau(i,u)]^{\alpha}[\eta(i,u)]^{\beta}}, & q < q_0(t); \\ \delta\{j - \arg\max_{u\in J(i)}[\tau(i,u)]^{\alpha}[\eta(i,u)]^{\beta}\}, & \text{otherwise}; \end{cases} \quad (10)$$

$$q_0(t) = 2[1 - 1/(1+\mathrm{e}^{-\lambda t})]. \quad (11)$$

其中:q 是$[0,1]$区间内的随机数,$q_0(t)$ 从 1 降低到 0;λ 是正参数,控制 $q_0(t)$ 的下降速度.

算法初期,$q_0(t)$ 较大,q 有较大概率小于 $q_0(t)$. 因此 P_{ij}^{k} 有很大概率选取式(10)的上半部分,即计算每条可选路径上的信息素与能见度的加权大小,然后按概率选择,体现了随机性. 算法后期,$q_0(t)$ 较小,q 有较小概率会小于 $q_0(t)$. 因此 P_{ij}^{k} 有很大概率选取式(10)的下半部分,即哪条路径上的信息素与能见度的加权最大,就选择哪条路径,体现了确定性(即随机性)的减弱.

5　实　　验

实验环境为 P4 1.85 GHz 处理器,512 MB 内存,Matlab 7.1. 随机生成 5 种不同点数的 DAG 图和相应的时间、面积、功耗、开发代价. 对比算法采用 ISA[3],ITS[4],IACA[5],DCG3A[6]. 算法参数设置见表 1.

表 1 AACA 关于节点数目的参数设置

节点	n	m	α	β	Q	γ	$\rho_{\min}$	λ
20	20	10	1	5.0	1000	0.95	0.10	0.06
40	40	20	1.2	4.0	800	0.95	0.10	0.06
60	60	30	1.1	3.5	2000	0.95	0.10	0.06
80	80	40	1.1	6.5	2500	0.95	0.10	0.06
100	100	50	1.2	7.5	1500	0.95	0.10	0.06

对于不敏感的参数,将其设置如下:n 必须与节点数目保持一致,m 选择节点数目的一半,$\gamma = 0.95$,$\rho_{\min} = 0.1$,$\lambda = 0.05$. 对于敏感参数,α,β 与 Q,通过实验选择最佳值.

5.1 运行时间比较

为公平起见,当这 5 种算法求解的结果相似时,比较运算时间. 结果列于表 2[3-6].

由表 2 可见,当 DAG 节点数极少时,AACA 略优,当节点数超过 40 时,AACA 效率明显优于其他 4 种算法. 节点数越多,AACA 的性能优势越强. 在节点规模高达 100 个时,AACA 消耗的时间仅有它们的 25 %～30 %. 图 3 给出了运算时间与节点规模的关系曲线.

表 2 不同算法运行时间对比(运行 100 次)

节点	ISA		ITS		IACA		DCG3A		本文算法	
	Time/s	Iterations	Time/s	Iterations	Time/s	Iterations	Time/s	Iterations	Time/s	Iterations
20	0.537	61.09	0.426	44.31	0.494	40.35	0.513	34.72	0.403	38.01
40	1.211	65.33	1.194	49.03	1.221	48.23	1.217	46.15	1.108	40.31
60	5.899	68.82	6.340	55.21	5.484	49.09	4.682	49.62	3.243	44.79
80	23.307	72.21	18.343	59.34	20.753	55.32	16.824	54.71	9.322	48.62
100	48.164	79.01	32.895	62.01	41.341	57.15	25.362	57.28	15.983	51.16

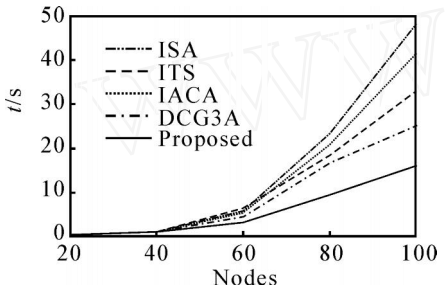


图 3 运算时间-节点规模曲线

5.2 收敛性能比较

比较这 5 种算法的收敛性能. 将每个算法运行 100 次,统计收敛到全局最优的次数,结果见表 3,部分数据摘自文献[3-6].

表 3 不同算法收敛性能对比(运行 100 次)

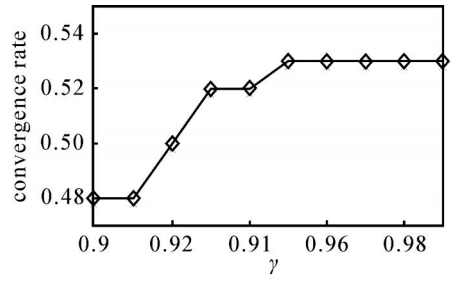
节点	ISA	ITS	IACA	DCG3A	本文算法
20	100	100	99	95	**100**
40	92	81	94	74	**92**
60	43	26	45	22	**75**
80	10	12	9	4	**53**
100	0	0	0	0	**12**

从表 3 可以看出,对于节点数目较少的系统,各个算法均能收敛到全局最优. 然而当节点数目较多时(>50),文献[3-6]的命中率就大幅下降,甚至无法收敛到全局最优;而本文算法的命中率虽然也随着节点数目的增加而降低,但是比[3-6]高得多. 例如,对于节点数目为 100 的系统,[3-6]的方法已经无法收敛到全局最优,然而本文方法仍然能够以 12 %的概率收敛.

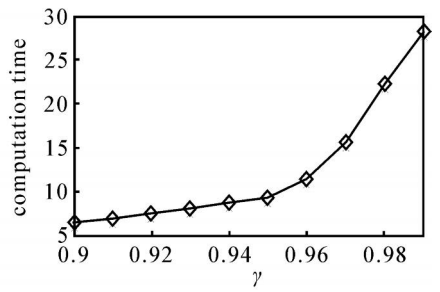
5.3 参数设置

关于 α,β,m,ρ,Q 的参数设置,文献[8]已全面叙述,故本文仅阐述与本算法有关的 γ,$\rho_{\min}$ 与 λ 以节点为 80 的 DAG 图为例(其余节点数的 DAG 图结论类似),令 γ 在[0.90,0.99]范围内变化,将结果示于图 4.

由图 4 可见,当 γ 过小,导致式(9)中的 $\rho(t)$ 下



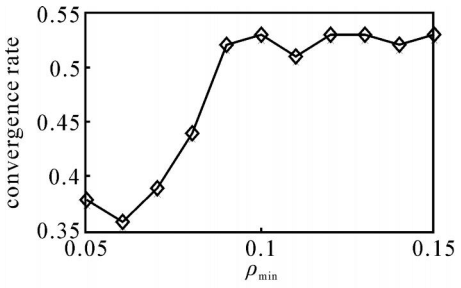
(a) γ对算法命中率的影响



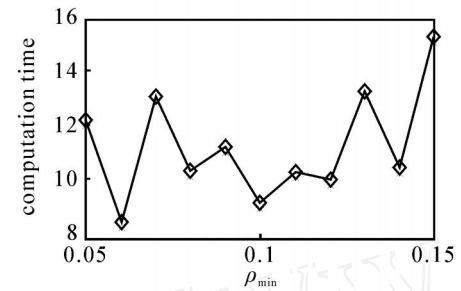
(b) γ对算法收敛时间的影响

图 4 γ对算法性能的影响

降过快,尽管算法收敛时间下降,但是更易收敛到局部最优;当 γ 过大,$\rho(t)$ 下降过慢,虽然能保证算法在最大程度上收敛到全局最优,但是付出的时间代价过大. 综合考虑,γ 取 0.95 较适宜. 将 γ 固定为 0.95, 再令 $\rho_{\min}$ 在 $[0.05, 0.15]$ 变动,结果示于图 5.



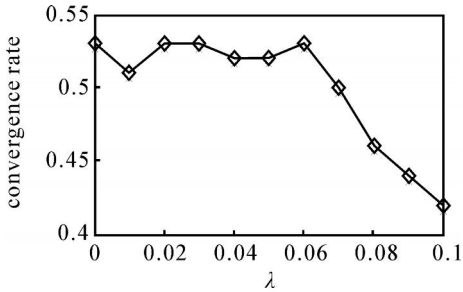
(a) $\rho_{\min}$对算法命中率的影响



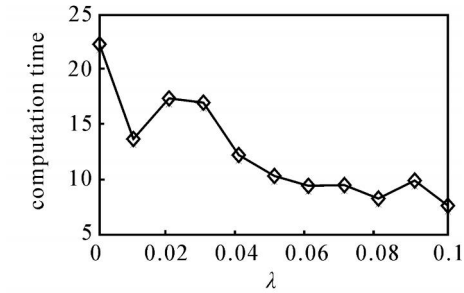
(b) $\rho_{\min}$对算法收敛时间的影响

图 5　$\rho_{\min}$ 对算法性能的影响

从图 5 可以看出,当 $\rho_{\min}$ 过小时,算法的命中率会大幅降低;当 $\rho_{\min}$ 过大时,算法在保持命中率不变的基础上,收敛时间在震荡中呈上升趋势. 综合考虑,$\rho_{\min}$ 取 0.1 较适宜. 最后,将 γ 固定为 0.95,$\rho_{\min}$ 固定为 0.01. 令 λ 在 $[0, 0.1]$ 之间变化,结果示于图 6.



(a) λ对算法命中率的影响



(b) λ对算法收敛时间的影响

图 6　λ对算法性能的影响

从图 6 可以看出,若 λ 过小,会导致式 (11) 的 $q_0(t)$ 下降过慢,则算法花费在全局搜索上的时间过多,导致算法收敛时间急剧增加;反之,则会导致 $q_0(t)$ 下降过快,没有充分进行全局搜索,使得收敛到局部解的概率增加. 综上,λ 取 0.06 较适宜.

6　结　　论

AACA 算法克服了传统 ACA 算法的不足,在嵌入式系统和 SoC 软硬件划分中取得了较好的效果,在耗费时间与命中率方面均优于文献[3-6]. 并且随着问题规模的增大,改进愈加明显.

参考文献(References)

[1] Ernst R, Henkel J, Benner T. Hardware-software cosynthesis for microcontrollers[J]. IEEE Design and Test of Computers, 1993, 10(4): 64-75.

[2] Saha D, Mitra R S, Basu A. Hardware software partitioning using genetic algorithm[C]. Proc of the 10th Int Conf on VLSI Design. Hyderabad: IEEE Computer Society Press, 1997: 155-160.

[3] Gajski D D, Vahid F, Narayan S, et al. SpecSyn: An environment supporting the specify-explore-refine paradigm for hardware/software system design[J]. Readings in Hardware/Software Co-design, 2002: 108-124.

[4] Vahid F, Stitt G. Hardware/software partitioning[J]. Reconfigurable Computing, 2008: 539-560.

[5] Wu J G, Srikanthan T. Low-complex dynamic programming algorithm for hardware/software partitioning[J]. Information Processing Letters, 2006, 98(2): 41-46.

[6] 熊志辉, 李思昆, 陈吉华. 遗传算法与蚂蚁算法动态融合的软硬件划分[J]. 软件学报, 2005, 16(4): 503-512.
(Xiong Z H, Li S K, Chen J H. Hardware/software partitioning based on dynamic combination of genetic algorithm and ant algorithm[J]. J of Software, 2005, 16(4): 503-512.)

[7] Kalvade A, Lee E A. The extended partitioning problem: Hardware/software mapping, scheduling and implementation-bin selection[J]. Design Automation of Embedded Systems, 1997, 2(1): 125-163.

[8] Duan H B, Wang D B, Yu X F. Research on the optimum configuration strategy for the adjustable parameters in ant colony algorithm[J]. J of Communication and Computer, 2005, 2(9): 32-35.