针对能耗热点的 SPM 静态分配管理策略

胡志刚,石金锋,蒋湘涛
HU Zhi-gang,SHI Jin-feng,JIANG Xiang-tao

中南大学 信息科学与工程学院,长沙 410083
School of Information Science and Engineering,Central South University,Changsha 410083,China
E-mail:himmea@163.com

HU Zhi-gang,SHI Jin-feng,JIANG Xiang-tao.Static allocation management strategy for SPM based on energy hotpot. Computer Engineering and Applications,2010,46(3):58-61.

Abstract: In this paper,a static management allocation strategy for scratchpad memory is proposed based on the energy hotpot of application.With instruction,data and global variable,this method transforms the application into a graph consisting of many distinct nodes based on WECFG,then computes the nodes access energy consumption and the nodes energy density,if nodes is placed in SPM,by SPM's average access energy consumption.It converts the Integer Iinear Programming(ILP) problem of SPM static allocation to a 0-1 backpack problem.Experiments show that this approach reduces about 34.8% energy consumption than that system without SPM.

Key words: energy hotpot;scratchpad memory;static allocation strategy

摘 要:综合考虑程序的指令块、数据块、全局变量对程序执行能耗的影响,使用带权重扩展控制流图(WECFG)将应用程序划分成各类逻辑节点,通过 SPM 平均访问能耗值计算出逻辑节点平均能耗,以及各逻辑节点的能耗密度。以能耗热点为依据构造 SPM 分配的整数线性规划算法(ILP),转化成以能耗密度为优先权的 0-1 背包算法。仿真结果表明,使用该分配策略的 SPM 空间分配,比不使用 SPM 时的能耗量平均减少 34.8%左右。

关键词:能耗热点;片上存储器;静态分配策略
DOI:10.3778/j.issn.1002-8331.2010.03.018 **文章编号:**1002-8331(2010)03-0058-04 **文献标识码:**A **中图分类号:**TP302

1 引言

片上存储器作为处理器和片外内存之间的一种缓存机制被引入到片上系统(System on Chip-SoC)。常用的片上存储器主要有两种:Cache 和 SPM(ScratchPad Memory)。

在片上系统性能不断提升的同时,对多数依靠电池来运行的片上系统而言,能耗成为片上系统发展的一大瓶颈。研究人员发现片上系统的内存子系统,占有整个片上系统的能量消耗总量约 50%~70%[1],因此,如何有效地降低内存子系统的能耗消耗就成为片上系统低功耗编译的重点。相对于硬件管理的 Cache,通过软件优化管理的 SPM,在性能、功耗和占用面积方面都优于 Cache[2];同时,SPM 不存在 Cache 命中/非命中的不确定性,使得 SPM 适用于对实时性要求更高的片上系统。以上特性让 SPM 成为未来片上系统中取代 Cache 的理想部件,然而不同于操作系统对 Cache 的自动管理,SPM 的空间管理必须在编译器的协助下人为进行。近年来研究人员提出了众多的 SPM 分配策略。

文献[2]提出了一种以全局变量为划分对象的 SPM 静态分配算法,算法以全局变量访问频率的高低在 SPM 和片外存储器之间分配变量,以达到降低系统能耗的目的,算法没有考虑程序中指令的分配。文献[3]将 SPM 分配问题转化为整数线性规划问题,所提出的分配方法只针对全局变量。文献[4]在对 SPM 进行静态分配时,同时考虑了指令块和全局变量的分配,没有考虑数据块。

上述文献所提出的 SPM 分配策略多数是将应用程序划分为存储在 SPM 的部分,以及存储在外存的部分,划分的对象主要是数据与指令,没有综合考虑应用程序的组成;SPM 分配依据是数据或者指令的访问频率,数据或指令的访问频率越高越有可能被放入 SPM,然而程序的数据部分虽然可能有很高的访问频率,但数据块所在空间可能很大,相对来说访问频繁的数据块的访问密度不一定很高,这就会影响分配算法的优化结果,原有分配算法对数据或者指令占用空间的大小没有预先考虑;分配过程通过整型线性规划解决器来完成,算法过程极其复杂。

提出一个基于能耗热点的 SPM 静态分配策略。该策略根

基金项目:国家自然科学基金(the National Natural Science Foundation of China under Grant No.60673165);湖南省自然科学基金(the Natural Science Foundation of Hunan Province of China under Grant No.07jj5077)。

作者简介:胡志刚(1963-),男,教授,主要研究领域为嵌入式系统低功耗编译,网格计算;石金锋(1983-),男,硕士生,主要研究领域为嵌入式系统低功耗编译;蒋湘涛(1980-),男,博士生,主要研究领域为嵌入式系统低功耗编译。

收稿日期:2008-10-07 **修回日期:**2008-12-29

据应用程序的执行时信息将其划分成带权重扩展控制流图 WECFG（Weighted Extend Control Flow Graph），WECFG 中的各个节点以及他们之间的调用关系描述了应用程序的执行流程；通过分析 WECFG 找出程序中的能耗热点，能耗热点既包括指令、数据，同时也包含全局变量；计算出 WECFG 中各个能耗热点的能耗密度，并提出一种以能耗密度为优先权的 SPM 0-1 背包分配算法。

2　基于能耗热点的静态分配策略

2.1　静态分配模型

图 1 所示为内存系统静态分配模型，阴影部分的 CPU core、Decoder、Cache 和 SPM 都属于片上存储器部分，白色部分 SDRAM 属于片外存储器。从片上存储器 SPM 访问数据或者指令所产生的能耗比从片外存取器 SDRAM 访问小得多。SPM 分配的目的就是将程序执行过程中访问能耗最高的部分从 SDRAM 植入到 SPM 空间（如图 1 的虚线右侧所示），以便降低程序执行过程中的访问能耗。在分配模型中 Cache 和 SPM 共存，这样做的目的是：其一，提出的针对能耗热点的静态分配算法是针对指令和数据同时有效的，对于没有放入 SPM 的指令或者数据，可通过 Cache 来访问；其二，便于对比有 SPM 和无 SPM 情况下的能量消耗变化。

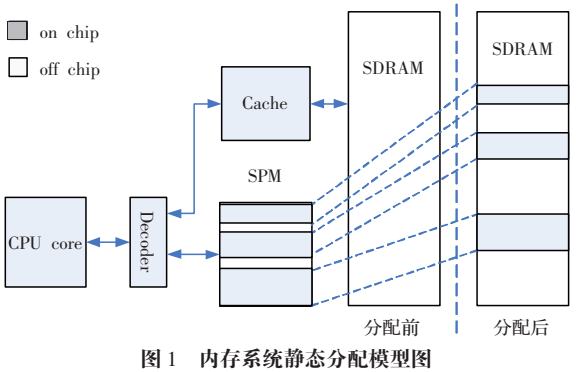

图 1　内存系统静态分配模型图

2.2　能耗热点的划分

应用程序在执行过程中，由于对不同指令和数据的访问频率不同，造成了整个程序执行时访问能量消耗不均匀，相对来说出现了一些能耗较高（访问较频繁）的节点块，这些节点块称作为能耗热点，如图 2 所示的节点 B_2 和 B_7。从图 2 中可以看出逻辑块 B_2 和 B_7 的执行频率远远大于程序的其他部分，因此所产生访问能耗也最高。

```
int array[100];                          B1
Proc1(){
    for(int i=0;i<100;i++){  } B2
        array[i]+=1;
    }
    Proc3();                             B3
}
Proc2(){
    …
    Proc3();                 } B4
    …
}
```

```
Proc3(){…}                               B5
void main()
{
    int localarray[100];                 B6
    for(int i=0;i<100;i++){  } B7
        localarray=i;
    }
    Proc1();                             B8
    Proc2();                             B9
}
```

图 2　应用程序的能耗热点

应用程序运行时的总能耗主要由两个部分组成：执行时能耗和访问时能耗，即：

$$E_{total}=E_{run}+E_{access} \quad (1)$$

执行时能耗的大小在特定体系结构的 SoC 中变化不大，多数时候是采用硬件的手段来降低执行时的能量消耗；访问能耗主要是处理器从内存子系统中访问数据和指令时的能量消耗，提出的 SPM 分配策略，就是通过在片外内存和 SPM 之间合理地安排数据和指令以达到降低应用程序的访问能耗。对于图2，SPM 分配算法的目的就是尽量将能耗热点 B_2 和 B_7 置入 SPM 空间，以降低程序执行时的访问能耗。

2.3　能耗密度定义

为了找到应用程序执行时的能耗热点，首先需要按照程序的执行流程将其划分成一系列的逻辑节点，文献[5]提出了一种扩展控制流图（Extend Control Flow Graph，ECFG），它将应用程序中所有符合大小限制的函数按照其内部的跳转指令划分成多个指令块，程序的全局堆栈被封装成全局数据变量。ECFG 将应用程序划分成全局数据变量、全局堆栈和函数指令块等节点，然而在计算程序的访问能耗热点时，ECFG 只考虑各个节点内部指令和数据所产生的能耗消耗，由于 ECFG 的各节点间存在调用或访问关系，因此一个节点的访问能耗大小也受到调用者访问次数的影响。在 EDFG 的基础上提出基于权重的扩展控制流图（Weighted Extend Control Flow Graph，WECFG），该图和 ECFG 的节点划分方式相同，所不同的是该图以权重方式给出了节点之间调用关系。WECFG 是一种带权有向图，图中的节点对应应用程序所划分成的逻辑单元，有向边表示节点对象之间的调用和访问等方向，边权重表示访问的属性信息以及访问的次数。例如图 3 为图 2 中代码的带权重扩展控制流图，程序从图 2 中 main 函数开始执行，main 函数中的局部变量数组对应图 3 中的 B_6 节点，紧接着的 for 循环对应图 3 中的 B_7 节点，节点 B_7 到节点 B_6 的有向边表示节点 B_7 对 B_6 访问，边上的 D:100 表示节点 B_7 访问节点 B_6 的数据 100 次。除了数据访问外，节点对象的访问属性可划分为表 1 所示的五种类型，如图 3 所示。其次，图中的各个节点也有其自身的权重，表示节点内部指令的执行次数。

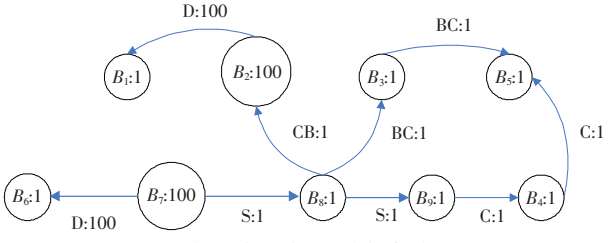

图 3　带权重扩展控制流图

表 1　对象间的访问类型

关系类型	关系符号	说明
$R_1(i,j)$	S	节点之间的顺序执行
$R_2(i,j)$	D	节点之间的数据访问
$R_3(i,j)$	C	节点之间的调用关系
$R_4(i,j)$	CB	节点之间的条件转移
$R_5(i,j)$	UB	节点之间的无条件转移

获得 WECFG 图以后，根据图中各个节点的划分，就可以计算各个节点从 SPM 获取指令或者数据时的能量消耗，如式(2)所示：

$$E_{spm}(B_i)=C_i*E_{spm}*[N_{read}(B_i)+N_{write}(B_i)] \quad (2)$$

式(2)中 E_{spm} 表示 SPM 平均访问能量消耗，$N_{read}(B_i)$ 和

$N_{write}(B_i)$表示节点 B_i 一次执行过程中读、写 SPM 的次数，C_i 表示节点 B_i 执行的次数，其中 C_i 是所有进去节点 B_i 的边的权重之和，及 WECFG 中相应节点进/出边的权重之和。

$$d(B_i)=\frac{E_{spm}(B_i)}{S(B_i)}=\frac{C_i*E_{spm}*[N_{read}(B_i)+N_{write}(B_i)]}{S(B_i)} \tag{3}$$

由式(2)访问能耗值除以节点体积得到节点能耗密度，如式(3)所示。式(3)中 $S(B_i)$表示节点 B_i 大小，以字节为单位。式(2)、(3)表示一种理想的情况，它表示节点从 SDRAM 搬移到 SPM 前后，大小和内容没有发生变化，实际上受到特定 SoC 系统跳转指令和数据装载指令的地址空间的限制，可能在原节点中引入额外的指令，这将导致了节点大小和内容的改变，内容的改变可能由于修改了节点中的指令或者向节点中加入了新的装载指令所引起。

计算节点能耗密度时，以节点位于 SPM 所产生的访问能耗为依据，考虑额外跳转指令的影响时，只考虑将节点置入 SPM 空间时的变化，不考虑位于 SDRAM 节点的变化。例如将图 2 中节点 B_7 置入 SPM，B_7 要访问位于 SDRAM 的 B_6 时，必须在 B_7 中加入额外的跳转指令以访问 B_6 节点，由于节点之间访问类型的不同，增加的跳转指令的数目和节点体积的变化可能不同，如表 2 所示。

表 2 对象间的关系

关系类型	添加跳转指令数 ΔN	节点体积变化 ΔS
$R_1(i,j)$	3	12
$R_2(i,j)$	3	12
$R_3(i,j)$	2	8
$R_4(i,j)$	2	8
$R_5(i,j)$	2	8

考虑引入额外指令对节点的影响，节点能耗密度为式(4)所示：

$$d(B_i)=\frac{E_{spm}(B_i)+\Delta E(B_i)}{S(B_i)+\Delta S(B_i)} \tag{4}$$

$$\Delta E(B_i)=C_i*E_{spm}*\sum_{h=1}^{5}\sum_{j=1}^{N}\Delta N(R_h(i,j)) \tag{5}$$

$$\Delta S(B_i)=\sum_{h=1}^{5}\sum_{j=1}^{N}\Delta S(R_h(i,j)) \tag{6}$$

$\Delta E(B_i)$和 $\Delta S(B_i)$表示将节点置入 SPM，插入的额外指令对能耗和节点体积的影响。

2.4 静态分配策略

通过 WECFG 分析得到各个节点访问能耗后，只要将程序中访问能耗值较高的节点放入 SPM，就可以有效降低程序的执行总能耗，但是，SPM 体积有限，通常情况下很小，因此，分配问题转化为：如何从程序中寻找能耗热点的最大独立集，最大独立集问题是一个 NP-complete 问题，可以将其转化为下列的整数线性规划(ILP)问题，首先定义下列的整型变量：

$$l_{spm}(B_i)=\begin{cases}1 & \text{hotpot } B_i \text{ is placed in SPM}\\ 0 & \text{otherwise}\end{cases} \tag{7}$$

$$l_{ext}(B_i)=\begin{cases}1 & \text{hotpot } B_i \text{ is placed in external memary}\\ 0 & \text{otherwise}\end{cases} \tag{8}$$

式(7)表示节点 B_i 放入 SPM，式(8)表示节点 B_i 存放在 SDRAM，对于任何节点 B_i，只能存放到在 SPM 或者 SDRAM 其一，因此必须满足式(9)的条件，其中 N 表示节点的总数目。建立以内存子系统访问能耗最小为目标的目标函数，如式(10)所示：

$$I_{spm}(B_i)+I_{ext}(B_i)=1 \text{ for all } B_i \ 0<i\leqslant N \tag{9}$$

$$E_{\min}(l_{spm},l_{ext})=\sum_{i=1}^{N}l_{spm}(B_i)*E_{spm}(B_i)+\sum_{i=1}^{N}l_{ext}(B_i)*E_{ext}(B_i) \tag{10}$$

式(10)中，$E_{spm}(B_i)$表示节点 B_i 从 SPM 访问时的能耗，$E_{ext}(B_i)$表示节点 B_i 从 SDRAM 访问时的能耗。式(10)在取得最小值时，即能耗热点的最大集被放置进了 SPM。该线性规划的约束条件是，放置到 SPM 的能耗块的体积必须小于 SPM 的空间大小，如式(11)所示：

$$\sum_{i=1}^{n}S(B_i)\leqslant S_{spm} \tag{11}$$

式(11)中的 n 表示放置到 SPM 中的节点数目。解决上述 ILP 问题时，可以选用 ILP 问题解决器来完成，但是 ILP 问题解决器寻找最高能耗节点集的过程极其复杂，并且得到的结果也并非最优解。将问题进行简化，通过对节点能耗密度的考虑，将分配问题简化为以能耗密度为优先级的 0-1 背包问题。

SPM 分配的目的是将访问能耗最高的节点置入 SPM，为此重新建立的目标函数如式(12)所示。从上述 ILP 问题分析过程中知道，SPM 分配时有两个影响因素需要考虑，分别是节点的访问能耗和 SPM 的容量，上述 ILP 问题是在程序中寻找访问能耗高且体积小的节点放入 SPM，即就是尽可能将能耗密度大的节点放入 SPM。将式(4)的代入式(12)中得到式(13)，以访问能耗为最大值的目标函数转化为以能耗密度为最大值的目标函数，当然，限制条件仍然是节点体积不能超过 SPM 空间大小，如表达式(14)所示。

$$E_{object}=\max\{E_{spm}(B_i)\} \tag{12}$$

$$E_{object}=\max\{E_{spm}(B_i)\}=\max\{d(B_i)*S(B_i)\}=\max\{d(B_i)\}*S(B_i)\ \ B_i\in L(B) \tag{13}$$

$$\sum_{i=1}^{n}S(B_i)\leqslant S_{spm} \tag{14}$$

根据式(4)计算出程序中节点的能耗密度值，按从大到小的顺序排序形成能耗密度链表 $L(B_i)$，SPM 分配转化为从 $L(B_i)$中选择一组访问能耗密度之和尽可能大的节点，将其放入 SPM，同时放入 SPM 的节点体积之和不能超过 SPM 空间大小。从表达式(13)可以看出，选择访问能耗最大节点的过程被转化为寻找能量密度最大的节点，这就降低了问题的求解难度，将复杂的整数线性规划问题，简化为以能耗密度为优先权 0-1 背包问题。选择能耗热点的算法如图 4 所示。

算法的 1~3 行首先将 WECFG 图中的各个逻辑节点的大小、平均访问能耗和能耗密度值存入数组 Size、Energy 和 L 中。算法 4~6 行对能耗热点 hotpot、非能耗热点 nonhotpot、能耗热点大小 sumSizeOfhotpot 进行初始化。7~11 行的 for 循环依次从数组 L 中选择能耗密度大的节点作为能耗热点，直到 SPM 空间填满为止，不能放入 SPM 的节点作为非能耗热点将被放入 SDRAM。得到能耗热点以后，SPM 分配时就可以将能耗热点置入 SPM 空间，以降低应用程序访问能耗。

3 仿真实验流程

3.1 能耗模型

根据图 1 中静态分配模型内存子系统的各个组成部分，以所有子系统的能量消耗来建立能耗模型，如式(11)所示，其中 E_{core} 表示 CPU core 的能耗，$E_{Decoder}$ 表示解析器能耗，E_{SPM} 表示 SPM 的能量消耗，E_{Icache} 与 E_{Dcache} 表示指令 Cache 和数据 Cache 的能量消耗，E_{ext} 表示 SDRAM 的能耗。

```
procedure HotpotChoose()
      //choose a maximal set of energy density node
      //from WECFG,and assign this set nodes in
      //SPM,ensure the sum size of the set node
      //less than or equal to the SPM size
1   put all nodes size get by analyzed
    WECFG in array Size[];
2   put all nodes energy consume get by analyzed
    WECFG in array Energy[];
3   calculate all nodes energy density,
    put density value in array L[] by decrease order;
4   hotpot={};
5   nonhotpot={};
6   sumSizeOfhotpot=0;
7   for(i=0;i<size of array L[];i++)
8       node←L[i];
9       if(sumSizeOfhotpot+Size[node]<Sspm)
10          hotpot←node;
11          sumSizeOfhotpot+=size[node];
12      else
13          nonhotpot←node;
    end procedure
```

图 4　能耗热点选择的 0-1 背包算法

$$E_{total}=E_{core}+E_{Decoder}+E_{SPM}+E_{Icache}+E_{Dcache}+E_{ext} \quad (11)$$

其中,各子部分的能耗如下:

$$E_{core}=t*P_{core} \quad (12)$$

$$E_{Decoder}=e_{TLB}*(hit+miss) \quad (13)$$

$$E_{SPM}=e_{SPM}*(read+write) \quad (14)$$

$$E_{Dcache}=E_{Icache}=e_{cache}*(hit+miss*linesize) \quad (15)$$

$$E_{ext}=E_{ext\_static}+E_{ext\_dynamic}=P_{standby}*\frac{coreclocks}{corefrequency}+e_{read_{random}}*read_{random}+e_{read_{burst}}*read_{burst}+e_{write_{random}}*write_{random}+e_{write_{burst}}*write_{burst} \quad (16)$$

Cache、SPM 以及 SDRAM 的能耗模型来自 CACTI[6]。仿真实验时,使用上述模型计算 SoC 内存子系统的各部分能耗值。

3.2　仿真流程

实验仿真流程如图 5 所示,经过编译的目标程序通过 WECFG 分析器被分解为目标节点链表,第 4 步对节点链表进行分析,分析得到各个节点的访问能耗和尺寸大小,从而计算出节点能耗密度,第 5 步依据对节点能耗分析和目标机器运行时信息,按照提出的 SPM 分配算法形成对节点的分配策略,第 6 步根据目标程序的分配策略和 SPM 以及其他系统部件的能耗模型在 SimpleScalar[7]模拟器中仿真程序的执行过程,第 7 步得出测试程序在此 SPM 结构下的能量消耗信息。

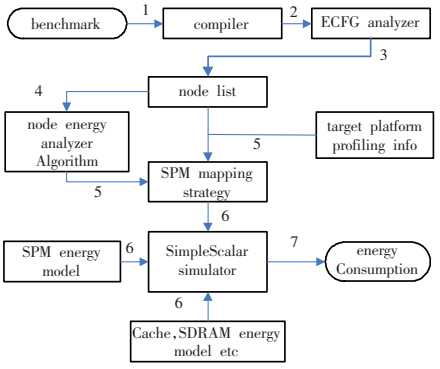

图 5　仿真流程图

3.3　仿真结果

实验中用到的 SPM、Cache 和 SDRAM 单位访问能耗的电气化参数如表 3 所示。

表 3　SPM、Cache 和 SDRAM 单位访问能耗

SPM read access(32 bit)	0.28 nJ
SPM write access(32 bit)	0.31 nJ
Cache read access(32 bit)	2.746 nJ
Cache write access(32 bit)	2.953 nJ
SDRAM single read access(32 bit)	40.563 nJ
SDRAM burst read access(32 bit)	51.341 nJ
SDRAM single write access(32 bit)	42.819 nJ

基准测试程序来自 Mediabench[8],实验结果如表 4 所示。

表 4　有无 SPM 的访问能耗比较

测试程序	S_c/byte	N_c(mJ)	N_p(mJ)	$1-N_c/N_p$(%)
rawcaudio	9 182	1 174	502	57.2
g721enc	11 052	2 932	1 659	43.4
mpeg2enc	26 808	3 876	2 817	27.3
pegwitdec	20 562	2 099	1 478	29.6
mpeg4enc	49 912	4 043	3 352	17.1
总计				34.8

表 4 中 S_c 表示测试程序占用的存储器容量,N_C 表示测试程序在 2KCache(8-way)+SDRAM 情况下的能耗大小,N_p 表示测试程序在 2KCache(8-way)+SDRAM+SPM 情况下的能量消耗。

从表 4 中可以看出,对比了有 SPM 和无 SPM 情况下,测试程序在 SimpleScalar 模拟器中的能量消耗。可以看出,通过将能耗密度大的节点放入 SPM 可以较大程度上降低系统能耗,能耗节约率约平均为 34.8%。内存子系统的能耗不仅受到有无 SPM 的影响,还受到 SPM 容量的影响,图 6 给出 g721enc 的能耗比较曲线。图中可以看出随着 SPM 容量的增加,SDRAM 能耗不断下降,SPM 能耗缓慢上升,内存子系统总能耗维持下降趋势,初期总能耗下降明显,后期下降逐渐平缓,这符合分配策略优先将能耗密度大的节点置入 SPM 的特点。同时应注意到,当 SPM 容量达到一定值时,总能耗基本不再下降,这是因为由增大 SPM 容量所带来的能耗收益,被维护 SPM 工作的能耗相抵消造成的。

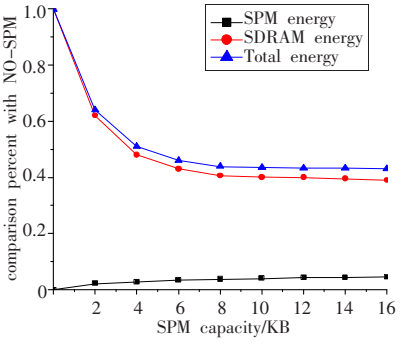

图 6　g721enc 能耗比较曲线

4　结束语

提出了一种基于能耗热点的 SPM 静态分配策略,使用扩展控制流图来划分和表示应用程序,在 SPM 分配过程中,以能耗密度为优先权将分配算法转化成相应的 0-1 背包问题。仿真

(下转 75 页)