热点感知的无线传感器网络数据存储策略

李巧勤[1] 吴 磊[2] 王 焱[1]
(电子科技大学计算机科学与工程学院 成都 611731)[1] (电子科技大学应用数学学院 成都 611731)[2]

摘 要 针对基于地理哈希表(GHT,Geographic Hash Table)的传感器网络数据中心存储(DCS,Data Centric Storage)机制的热点问题,提出了能量有效的热点感知数据存储策略SASS(hotSpot-Aware data Storage Strategy),对GHT的路由策略进行改进,以减少边界模式引起的能量消耗,并利用邻居节点动态地扩展存储空间。仿真结果表明,与现有存储策略相比,SASS能有效减少因存储资源限制引起的数据丢失,并减轻热点区域节点的通信负载。

关键词 无线传感器网络,数据中心存储,存储热点,事件
中图法分类号 TP393 **文献标识码** A

Hotspot-aware Data Storage Strategy for Wireless Sensor Networks

LI Qiao-qin[1] WU Lei[2] WANG Yan[1]
(School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)[1]
(School of Applied Mathematics, University of Electronic Science and Technology of China, Chengdu 611731, China)[2]

Abstract Considering hotspot problem in GHT(Geographic Hash Table)-based Data-centric storage (DCS) sensor networks, this paper proposed a hotspot-aware data storage scheme (SASS), which improves the routing scheme of GHT to reduce the energy consumption caused by perimeter walk, and exploits neighboring nodes to extend storage space dynamically. Simulation results demonstrate that compared with exiting schemes, SASS can effectively alleviate data lost caused by the limitation of storage space and reduce communication load of nodes in hotspot areas.

Keywords Wireless sensor networks, Data-centric storage, Storage hotspot, Events

1 引言

传统的无线传感器网络(WSN)通过节点之间的相互协作将数据传送到数据汇聚节点[1]。在一些特定的WSN应用中,用户可能只对某些特定事件感兴趣。如果将大量原始数据传送到汇聚点,会造成不必要的资源浪费,因此需要实现网络内的数据处理[2]。以数据为中心的存储(DCS)机制[3]为这类应用提供了解决方案。在基于DCS的传感器网络中,以数据名字或类型为依据在网络内选择存储节点。传感器节点监测到特定类型的数据或事件时,将其发送到对应的存储节点。对该类数据或事件有兴趣的节点向对应的存储节点发送查询请求,以获取相应信息。

地理哈希表GHT[4]是DCS的典型代表,其根据数据类型产生存储位置,并利用GPSR[5]将数据转发到位于存储位置的节点,即主存储节点(home node)。若没有节点位于存储位置,则距离存储位置最近的节点成为主存储节点。在这种情况下,数据被主存储节点接收之前,需要围绕存储位置环绕一周,所经过节点之间的路径构成存储位置的主边界(home perimeter)。当某个节点需要查询该类型的数据时,通过同样的映射机制得到该类数据的存储位置,并将查询请求转发到存储位置。相同类型的数据被映射到相同的存储位置,如果同类事件过多,则会造成主存储节点有较大的负载,产生存储热点问题。为了缓解存储热点问题,GHT提出SR(Structured Replication),每类事件除了一个根(root)存储节点之外,还有4^d-1个镜像(mirror)存储节点,其中d为层次深度。节点产生事件时,从所有4^d个存储节点中选择最近的节点存储。查询请求首先发送到root节点,然后按层次结构逐层发送到每一级的镜像存储节点。若事件在网络区域均匀分布,SR可减少单个存储节点的负载;但如果事件在网络区域内分布不均匀,则多个存储节点并不能有效对负载分流,还会因为查询请求需要发送多个存储节点而增加查询开销。因此SR并不适合事件分布不均匀的场合。GHT的另一个问题是在出现空存储位置(即存储位置处没有节点)时进入边界(perimeter)模式,造成大量能量浪费,进一步加重了热点区域节点的通信负载。

针对GHT的问题,相关研究提出了改进措施。文献[6]考虑节点非均匀分布的场合,提出根据节点分布和数据的重要程度来确定每类数据的存储节点数量。文献[7]提出多阈值Cover-Up-Face策略,其可以动态扩展存储节点,将事件分散到多个节点,以实现节点之间的能量平衡消耗。该策略可

到稿日期:2012-06-19 返修日期:2012-10-11 本文受国家自然科学基金(61103208),中央高校基金(ZYGX2010J111,ZYGX2011J102,ZYGX2011J058)资助。

李巧勤(1972-),女,博士,讲师,主要研究方向为无线传感器网络、机会网络,E-mail:helenli803@163.com;吴 磊(1978-),男,博士,讲师,主要研究方向为移动传感器网络、车载网络;王 焱(1977-),女,副教授,主要研究方向为计算机网络。

以缓解热点区域的数据丢失，但数据仍需通过原来的主存储节点转发到新的存储节点，加重了主存储节点的负载。在主边界上可能有多个节点存储同类数据，查询请求需要在主边界上环绕一周，不能避免边界模式引起的能量消耗。文献[8]提出优化数据存储 ODS (Optimal Data Storage)策略，为每类事件指定一个存储节点，同一类事件的存储节点可以根据产生事件节点的地理位置或者事件产生和查询频率而自动改变，使网络总能量消耗最小，并减少查询延迟。在有多个事件源节点和查询节点的情况下，为了降低计算复杂度，提出近似优化策略 NDS(Near-optimal Data Storage)代替 ODS 策略。这两种策略都可以最小化网络的总能量消耗，但没有考虑节点之间能量和存储负载的不均衡问题。文献[9]提出 Tug-of-War (ToW)，根据事件频率和查询频率动态地调整每类事件的主存储节点个数，以减小网络的总代价。然而，在有些应用场合中，事先并不一定知道查询频率和查询节点的位置。

针对 GHT 的存储热点问题，本文提出能量有效的热点感知存储策略 SASS。与以上文献中提出的方法相比，SASS 主要有以下不同：1)根据存储节点的空间使用情况动态进行存储扩展，而现有方法大多是对所有类型的事件都分配同样的存储空间，即指定相同的存储节点个数；2)在存储扩展之后，事件可以直接转发到新的存储节点，而不必经过主存储节点的转发，可以减小主存储节点的通信开销；3)出现空存储位置的情况下，可以有效减小边界模式引起的通信开销。

2 热点感知的存储策略设计

2.1 网络模型

假设所有传感器节点随机均匀分布在 $R\times R$ 的二维矩形区域内，并具有如下属性：(1)所有节点都静止；(2)节点有相同的通信范围；(3)节点密度足够大，可以保证节点在各个方向上都有很好的连通性；(4)所有节点知道网络的边界，并可通过相关技术获知节点的位置信息[10]，因为在一些特定应用中，只有知道节点的位置信息，数据对用户才是有效的，比如目标跟踪，因此该假设是合理的；(5)传感器网络通过 sink 节点与外界连接。

事件的产生是随机的，节点监测到事件时，根据事件类型，通过哈希函数生成事件的存储地址 L，距离 L 最近的节点成为该类事件的主存储节点。将网络划分为等面积的网格，假设位于同一网格区域内的节点产生相同类型的事件。每个网格有一个二维编号 (X,Y)，网络区域左下角坐标为 (x_0,y_0)，网格宽度为 d，如图 1 所示。每个节点根据自己所在位置 (x_i,y_i) 可以计算所在网格区域的编号：

$$\begin{cases} X=\lfloor (x_i-x_0)/d \rfloor \\ Y=\lfloor (y_i-y_0)/d \rfloor \end{cases} \tag{1}$$

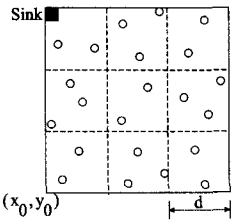


图 1 网格区域示意图

2.2 事件存储

节点产生事件时，按预定义的哈希函数计算存储位置 L，并将分组转发至距离 L 最近的节点。为了减少边界模式的能量消耗，本文对 GHT 的路由策略进行改进，以减少热点区域节点的能量消耗。每个节点维护一个列表 CS_List，用于记录事件的存储位置(L)、主存储节点位置(H_S)和当前存储节点位置(C_S)之间的对应关系，网络初始化时 CS_List 为空。主存储节点第一次接收事件后，向邻居节点广播消息，包括该类事件的 L、H_S 和 C_S 等信息。邻居节点收到主存储节点的广播消息，在 CS_List 中记录相应的信息。进行存储扩展之前，事件的当前存储节点为主存储节点，即 $H_S=C_S$。

假设节点接收到目的地址为 L 的分组，改进后的转发过程描述如下：

Step1 首先判断节点的 CS_List 是否为空，若为空，则执行 Step4，否则继续执行；

Step2 以事件的存储位置 L 为关键字在 CS_List 中查找，若未找到，则执行 Step4，否则继续执行；

Step3 若 $C_S\neq H_S$，则用当前存储节点位置(C_S)替换事件的存储位置(L)；

Step4 转发到下一节点。

按照上述过程执行事件转发，事件到达当前存储节点的邻居节点时，将事件的存储地址修改为当前存储节点地址，则事件到达存储节点后直接被接收，而不再以边界模式环绕存储位置，这将减少存储节点及主边界上节点的能量消耗。改进后的转发过程伪代码实现如图 2 所示。

```
if (CS_List is not NULL)              //step1
  if (found Lin CS_List)              //step2
    if (Cs≠Hs)                        //step3,storage extended
      replance L with Cs;             //modify destination
    end if
  end if
end if
send to next nodel;                   //step4
```

图 2 转发过程伪代码实现

2.3 存储扩展

为了减少事件丢失，当主存储节点的存储空间降到门限值时，利用邻居节点进行存储扩展。存储扩展时除了考虑候选节点的可用存储空间之外，还需考虑节点的剩余能量。在介绍存储扩展过程之前，先引入节点生存时间(T)的计算方法。该参数取决于可用存储时间(T^s)和可用能量时间(T^e)中的较小者，其中，T^s 表示按照当前事件到达率，节点存储达到饱和的时间；T^e 表示按当前事件到达率，节点耗尽能量的时间。节点 i 在时刻 t 的 T^s、T^e 和 T 分别计算如下：

$$T_i^s(t)=\frac{S_i(t)}{\lambda(t)} \tag{2}$$

$$T_i^e(t)=\frac{E_i(t)}{E(\lambda(t))} \tag{3}$$

$$T_i(t)=\mathrm{Min}(T_i^s(t),T_i^e(t)) \tag{4}$$

式中，$S_i(t)$ 为节点 i 的可用存储空间；$\lambda(t)$ 为事件到达率；$E_i(t)$ 表示节点剩余能量；$E(\lambda(t))$ 表示在事件到达率为 λ 的情况下，节点单位时间内接收事件消耗的能量。这里计算 T^e 时只考虑了节点成为新的存储节点时接收该类事件的能量消耗，但实际上节点还可能有其他情况的消耗能量，比如接收查询请求和发送查询结果等，因此式(3)只考虑了节点能量消耗的下限。根据上述参量，每个节点计算作为候选扩展节点的

优先值 P。这里将节点 i 的优先值 P_i 定义为三元组 $P_i=\langle T_i,B_i,d_i\rangle$，其中，$B_i$ 反映 T^s 和 T^e 之间的大小关系，定义为 $B_i=\frac{T^e-T^s}{T^s}$；d_i 为节点与存储位置 L 之间的距离。以上所有的计算都是在节点接收到存储扩展请求时才进行。

基于能量代价的因素，存储扩展节点只在主存储节点的邻居节点中选择，通过 3 次握手实现。主存储节点向其邻居节点广播 Etd_Req 消息，包括事件的存储位置、主存储节点位置和当前事件到达率 $\lambda(t)$ 等，请求存储扩展。邻居节点收到 Etd_Req 消息后，根据 $\lambda(t)$、剩余能量和存储空间等信息计算 T^s、T^e、T 和 P，然后向主存储节点发送 Etd_Resp 消息，包括节点 ID、节点位置、节点优先值 P 等。主存储节点从所有 Etd_Resp 消息中选择优先值 P 最大的节点作为扩展存储节点，并向其发送 Etd_Ack 消息。收到 Etd_Ack 的节点成为扩展存储节点。

存储扩展之后，为了实现事件直接发送至扩展存储节点，在通过 3 次握手完成扩展存储节点的选择之后，主存储节点和扩展存储节点分别在各自的通信范围内广播 S_Change 消息，包含〈存储位置，主存储节点地址，扩展存储节点地址〉。收到 S_Change 消息的每个节点在列表 CS_List 中以存储位置 L 为关键字进行查找，若找到，则将 C_S 修改为当前扩展存储节点地址；若没有找到，则在 CS_List 中增加一条记录，以反映当前存储节点的变化。节点转发该类事件时，将目的地址修改为当前存储节点地址，事件可以不经过主存储节点而直接转发到扩展存储节点，这有利于减少边界模式引起的能量消耗。假设 E_N 为网络中总的事件类型数，ρ 为平均邻居节点个数，则列表 CS_List 的长度上限为 $\mathrm{Min}\{E_N,\rho\}$。

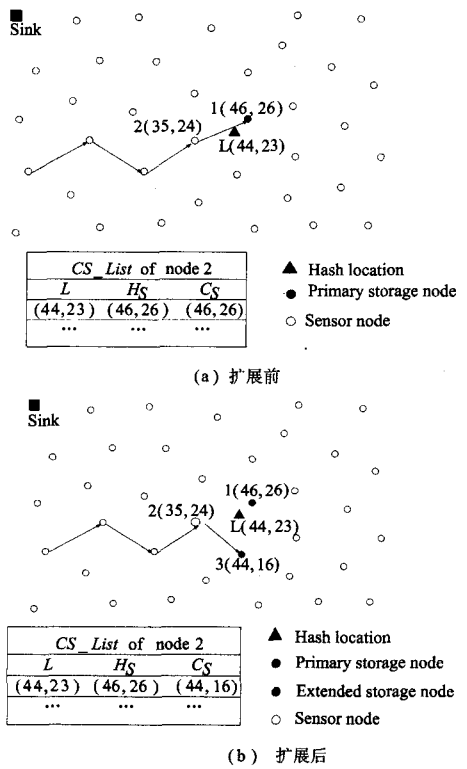


图 3 存储扩展示意图

图 3 为存储扩展示意图。如图 3(a)所示，源节点产生的事件映射到位置 $L(44,23)$。存储扩展之前，事件路由到距离 L 最近的主存储节点 1(46,26)。当节点 1 存储空间降到门限值时，按存储扩展算法选择节点 3 为扩展存储节点，如图 3(b)所示。节点 2 转发事件时查询 CS_List，并将事件目的地址修改为当前扩展存储节点 3 的地址(44,16)，事件经节点 2 直接被转发到扩展存储节点 3，而不需要经过主存储节点 1 的转发，减少了节点 1 的能量开销，也避免了边界转发。

若当前扩展存储节点的存储空间或能量降到门限值，则在通信范围内广播 Etd_Cancel()消息，包括〈存储位置，主存储节点地址，扩展存储节点地址〉等。主存储节点收到 Etd_Cancel()消息，则广播该信息通知邻居节点，并通过 3 次握手重新选择新的扩展存储节点。若通信范围内不能找到新的扩展存储节点，事件仍被发送到主存储节点，并依据 FIFO 原则丢弃事件。

存储扩展过程的伪代码如图 4 所示。

```
switch (message_type)
  case Etd_Req: //storage extend request
    compute Te, Ts, T, P;
    send (Etd_Resp) to HN;
    break;
  case Etd_Ack: //selected as extended storage
    broadcast (S_Change)
    break;
  case S_Change: //storage extended, destination L
    if (not found L in CS_List)
      insert (L, HN_location, EN_location) to CS_List;
    else
      update CS with EN_location;
    end if
    break;
  case Etd_cancel: //destination L
    if (found L in CS_List)
      update CS with HS;
    end if
    break;
end switch;
```

图 4 存储扩展过程伪代码

3 性能分析

本节对 SASS 的能量效率进行分析。事件存储过程中的能量消耗与事件插入路径长度有关，因此这里对 SASS 和 GHT 的平均事件插入路径长度进行比较，以分析 SASS 的能量代价。当节点密度足够大时，边界模式主要由空存储位置引起。GHT 的平均事件插入路径长度 L_G 可以分为两部分：贪婪模式的路径长度(g)和边界模式的路径长度(p)。在 $R\times R$ 的矩形区域，假设节点通信半径为 r，由文献[11]可得：

$$g=\left(\frac{2+\sqrt{2}+5\ln(1+\sqrt{2})}{15}\right)\frac{R}{r} \tag{5}$$

用 ρ 表示平均节点密度(邻居节点数)，N 表示网络节点总数，则有 $\frac{R}{r}=\sqrt{\frac{N\pi}{\rho}}$，代入式(5)，可得：

$$g=\left(\frac{2+\sqrt{2}+5\ln(1+\sqrt{2})}{15}\right)\sqrt{\frac{N\pi}{\rho}} \tag{6}$$

边界模式下的路径长度 p 取决于包含边界路径的平面图表面大小，由于篇幅限制，这里略去 p 的详细推导过程。在有限网络的 Gabriel 图(GG)[12]情况下，可得：

$$p=\frac{4(1-e^{-\frac{\rho}{4}(1-\frac{2}{3\pi})})}{1+\frac{2}{N}-2e^{-\frac{\rho}{4}(1-\frac{2}{3\pi})}} \tag{7}$$

SASS策略可以避免在空存储位置情况下的边界转发，因此其平均事件插入路径长度 $L_S \approx g$。与 GHT 相比，其归一化代价为：

$$\eta=\frac{L_S}{L_G}=\frac{g}{g+p} \tag{8}$$

结合式(6)—式(8)，可知 η 与 ρ 和 N 有关。图 5 显示 η 随节点密度的变化趋势。节点密度为 10 时，η 的值达到最大。随着节点密度的增加，边界模式的长度趋于稳定值，η 主要取决于网络规模。在节点密度相同的情况下，随着节点数的增大，g 的增长速度大于 p，因此 η 随节点数的增加而增大。

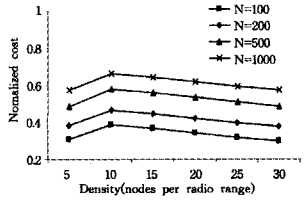


图 5　SASS 的归一化事件插入代价

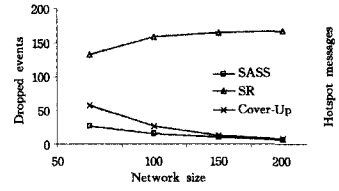
4　仿真实验

为了验证 SASS 的性能，利用 OPNET 模拟实现了 SASS 策略，并与 SR[4] 和 Cover-Up-Face[7]（简称 Cover-Up）的性能进行了比较。其中，Cover-Up 的存储门限级别设置为 1，SR 的层次深度设置为 1，即 $d=1$。

采用第 2.1 节描述的网络模型，随机选择 20 个节点作为事件源节点产生事件，每个源节点按泊松分布到达过程产生事件，前 50s 为网络初始化时间，完成初始化之后源节点开始产生事件。为了模拟事件分布不均匀的场景，其中两个源节点每 1s 产生 1 个事件，其余的源节点每 5s 产生 1 个事件，150s 时所有源节点停止产生事件。之后，sink 节点开始进行事件查询，查询间隔为 5s，200s 时仿真结束。所有节点的存储空间为 50 单位（以事件为单位）。以下所有仿真数据都是 10 次仿真结果的平均值。

第一组实验的节点数在 50 到 200 之间变化，改变网络区域大小、保持节点密度（平均邻居节点）不变，研究网络规模（节点数）对性能的影响。事件丢失随网络规模的变化趋势如图 6 所示。SASS 和 Cover-Up 都在主存储节点存储空间用完时选择新的节点进行存储，事件丢失数量随网络规模的增大而减少。因为随着网络规模的增大，在事件类型相同的情况下，主存储节点相对分散，出现存储热点时有更多的候选节点可用于存储扩展。SR 选择最近的存储节点进行存储，但对于事件分布不均匀的情况，同类事件并不能均匀分布到 4 个存储节点，而是集中在靠近事件源的存储节点，因此并不能充分利用 mirror 存储减少单个存储节点的负载，事件丢失明显高于其他两种策略。

图 7 显示节点的最大负载随网络规模的变化趋势。总体来看，所有策略的节点最大负载随网络节点数的增加而下降。节点最大负载出现在热点区域，随着节点数的增加，不同事件的存储位置相对分散，热点区域节点转发其他类型事件的数量减少，通信负载下降。在网络节点数相同的情况下，SASS 的最大负载在大部分场合小于其他两种策略，一方面 SASS 减少了边界模式的能量开销；另一方面，在存储扩展后，直接将事件转发到当前的扩展存储节点，而不必经过主存储节点的转发，有效减少了热点区域节点的通信负载。SASS 在网络规模较小时对热点区域节点负载的改善更显著。随着网络规模的增大，SR 的多个存储节点的优势有助于减少热点区域的节点最大负载。



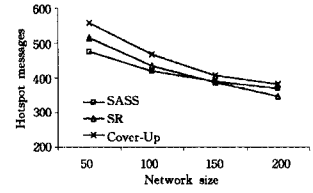
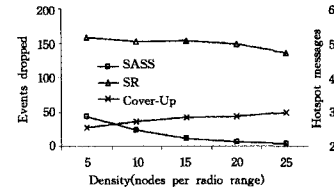


图 6　网络规模对事件丢失的影响　图 7　网络规模对最大负载的影响

第二组实验的网络区域大小固定，通过改变节点数量来改变节点密度，研究节点密度对性能的影响。图 8 显示事件丢失随节点密度的变化趋势，3 种策略的变化趋势有所不同。随着节点密度的增加，SR 的事件丢失没有明显变化；Cover-Up 在主边界上选择新的存储节点，而主边界的长度（节点数）随节点密度的增加呈下降趋势，因此候选存储节点数减少，事件丢失随节点密度的增加呈上升趋势；SASS 的事件丢失随节点密度的增加呈下降趋势，因为候选存储节点的数量随节点密度的增加而增加。在节点密度相同的情况下，SASS 在多数场合（density>5）的事件丢失数小于其他两种策略，随着节点密度的增加，其优势越明显。

节点最大负载随节点密度的变化趋势如图 9 所示。在节点密度较小时，3 种策略的最大节点负载随节点密度的增加有所下降，这是因为在网络区域固定的情况下，节点总数随着节点密度的增加而增加，有更多的节点分担负载，存储节点转发其他类型事件的概率减小，因此节点的最大负载减小。随着节点密度进一步增加，节点最大负载趋于稳定，进一步增加节点密度对最大负载没有明显改善。在节点密度相同的情况下，SASS 因为避免了边界转发，有助于减少热点区域节点的最大负载。



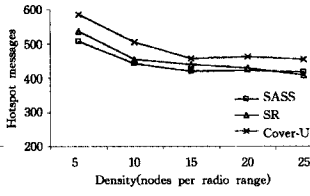


图 8　节点密度对事件丢失的影响　图 9　节点密度对最大负载的影响

结束语　基于 GHT 的传感器网络数据中心存储机制将同类事件映射到相同的存储节点，事件分布不均匀时会产生存储热点问题。本文提出热点感知的数据存储策略 SASS，以有效缓解热点区域的存储和通信瓶颈。SASS 首先对路由策略进行改进，以减少边界模式的能量消耗，并在此基础上基于存储节点的存储空间使用情况动态选择邻居节点进行存储扩展，以减少事件丢失。仿真实验表明，与结构化复制方法 SR 和已有的一种存储热点解决方法 Cover-Up 相比，SASS 能有效缓解存储热点问题，减少事件丢失和热点区域节点的能量开销。对于下一步研究工作，我们将考虑采取区域存储的方法来更有效地解决传感器网络的存储热点问题。

参 考 文 献

[1] 赵忠华,皇甫伟,孙利民,等. 无线传感器网络管理技术[J]. 计算机科学,2011,38(1):8-14

[2] Xiang Qiao, Zhang Hong-wei, Xu Jin-hong, et al. When in-networks processing meets time: Complexity and effects of joint optimization in wireless sensor networks[J]. IEEE transactions on mobile computing, 2011, 10(10): 1488-1502

[3] Shenker S, Ratnasamy S, Karp B, et al. Data-centric storage in sensornets[J]. SIGCOMM Computer Communication Review, 2003, 33(1): 137-142

[4] Ratnasamy S, Karp B, Shenker S, et al. Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table[J]. Mobile Networks and Applications, 2003, 8(4): 427-442

[5] Karp B, Kung K. GPSR: greedy perimeter stateless routing for wireless networks[C]// Proc. of the 6th annual international conference on mobile computing and networking. ACM: New York, USA, 2000: 243-254

[6] Michele A, Stefano C, Francesco N, et al. Dealing with non uniformity in data centric storage for wireless sensor networks[J]. IEEE Transactions on Parallel and Distributed System, 2011, 22(8): 1398-1406

[7] Liao Wen-hwa, Wu Wan-chi. Effective hotspot storage management schemes in wireless sensor networks[J]. Computer Communications, 2008, 31: 2131-2141

[8] Yu Zhao-chun, Xiao Bin, Zhou Shui-geng. Achieving optimal data storage position in wireless sensor networks[J]. Computer Communications, 2010, 33: 92-102

[9] Joung Y-J, Huang S-H, Lin Shi-hang. Making data-centric storage adaptive and cost-optimal[J]. Computer Networks, 2012, 56(1): 213-230

[10] Das S K, Wang Jing, Ghpsh R K, et al. Theoretical aspects of distributed computing in sensor networks[M]. Springer, 2011: 257-291

[11] Weisstein E W. Square line picking[OL]. http://mathworld.wolfram.com/SquareLinePicking.html

[12] Bratislav M, Miroslaw M. Dropped edges and faces' size in Gabriel and relative neighborhood graphs[C]// Proc. of IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS). 2006: 407-416

(上接第 58 页)

但对于开发指令集并行度而言,基本块数量的减少、单个基本块范围的扩大,有助于指令调度开发指令级并行。

表 1 if 转换算法改进前 main.c 的运行情况

分支方向(a=?)	-5	1	5
执行周期(拍)	6582	7877	7865
执行包(个)	46	46	41

表 2 if 转换算法改进后 main.c 的运行情况

分支方向(a=?)	-5	1	5
执行周期(拍)	1978	1978	1978
执行包(个)	21	21	21

结束语 编译器对计算机性能的开发起着至关重要的作用,特别是有指令集并行特征的处理器对编译器的要求更高。为了提高编译器性能,充分利用硬件资源,编译器中嵌入了很多优化算法,而指令调度又是关键所在。谓词执行技术的应用能合并基本块,消除控制流中的分支,从而为指令调度级的优化提供更加广阔的调度空间,能充分挖掘指令级并行度。研究表明,有效地消除分支可以使指令级并行度提高到原来的 3 倍,而谓词执行技术能够平均消除 56% 的分支错误和 27% 的分支[10,11]。本文以 GCC 内部的 if 转换算法为基础,移植并改进了 Matrix 编译器中的 if 转换功能和算法,成功地实现了 Matrix 编译器的 if 转换算法。实验结果表明,本文设计实现的 if 转换有效地消除了测试程序中的分支,增加了调度的基本块的范围,简化了控制流图,有助于编译器生成更加优化的代码。

但是在选择符合 if 转换的 HyperBlock 组合时,并没有考虑基本块的执行频率、分支可能性、基本块大小等信息,这样一些执行频率低、分支可能性小的基本块在 if 转换后反而会使代码的执行效率变低。所以本文的下一步工作是,根据 GCC 内部自带的 profile 信息[4]、执行频率和基本块大小等信息选择符合 if 转换的 HyperBlock 组合。

参 考 文 献

[1] 张晨曦,王志英,张春元,等. 计算机体系结构(第二版)[M]. 北京:高等教育出版社,2005: 79-80

[2] 田祖伟,赵克佳,汪小飞. GCC 基于 IA-64 谓词执行的 if 转换技术研究[J]. 微电子学与计算机,2005,22(6): 188-196

[3] Strätling A. Optimizing the GCC Suite for a VLIW Architecture [OL]. http://www.qucosa.de, 2013

[4] Stallman R M. GNU Compiler Collection (GCC) Internals[OL]. http://gcc.gnu.org, 2013

[5] Kumar R, Saxena A K, Singh P K. A Novel Heuristic for Selection of HyperBlock in If-Conversion[J]. Electronics Computer Technology, 2001, 6: 232-235

[6] Jacome M F, de Veciana G, Pillai S. Clustered VLIW Architectures with Predicated Switching[A]// DAC '01 Proceedings of the 38th annual Design Automation Conference, 2001[C]. New York: Association for Computing Machinery, 2001: 696-701

[7] Park J C H, Schlansker M. On Predicated Execution. Software and Systems Laboratory[OL]. https://www.hpl.hp.com, 2013

[8] Zimmerman E, Zilles C. On the Energy Effectiveness of If-conversion in Superscalar Microprocessors[OL]. http://www.cs.illinois.edu, 2013

[9] Chuang Wei-haw, Calder B, Ferrante J. Phi-Predication for Light-Weight if-Conversion[A]// Proceedings of the International Symposium on Code Generation and Optimization: Code Generation and Optimization, CGO 2003. International Symposium on, 2003[C]. California, 2003: 179-190

[10] Monica S L, Robert P W. Limits of control flow on parallelism [A]// ISCA '92 Proceedings of the 19th annual international symposium on Computer architecture, 1992[C]. New York: Association for Computing Machinery, 1992: 46-57

[11] Mahlke S A, Lin D C, Chen W Y, et al. Effective compiler support for predicated execution using the HyperBlock[A]// MICRO 25 Proceedings of the 25th Annual International Symposium on Microarchitecture, 1992[C]. New York: Association for Computing Machinery, 1992: 45-54