第35卷　第17期　　计　算　机　工　程　　2009年9月
Vol.35　No.17　　Computer Engineering　　September 2009
·开发研究与设计技术·　文章编号：1000—3428(2009)17—0241—03　文献标识码：A　中图分类号：TP301.6

并行DSP系统消息传递路由算法

王　哲，王希敏
(海军工程大学电子工程学院，武汉 430033)

摘　要： 为了提高DSP系统软件的移植性，设计消息传递路由算法。采用邻接表存储并行系统硬件拓扑结构，增加节点数据流信息为算法搜索的限制条件以提高算法效率。以ADSPTS101并行系统为例，使用VisualDSP++平台实现并验证该算法。结果表明，该算法有效解决并行DSP系统的消息传递问题，提高系统性能，在并行DSP系统中有较强通用性。

关键词： 并行DSP系统；消息传递；路由算法；数据流

Parallel DSP System Message-passing Routing Algorithm

WANG Zhe, WANG Xi-min
(College of Electronic Engineering, Naval University of Engineering, Wuhan 430033)

【Abstract】 In order to improve the portability of the parallel Digital Signal Processing(DSP) system, this paper designs a message-passing routing algorithm. It advances the efficiency of algorithm through using adjacency list to store the structure of system hardware topology and enhances information of node data flows as restrictive qualification for algorithm searching. ADSPTS101 parallel system is as an example, it implements and validates the routing algorithm using VisualDSP++ platform. Result shows that the algorithm solves the problem of message-passing in parallel DSP system effectively, enhances the performance of system, and it has more applicability in parallel DSP system.

【Key words】 parallel Digital Signal Processing(DSP) system; message-passing; routing algorithm; data flow

1　概述

随着科学技术的发展，单片数字信号处理(Digital Signal Processing, DSP)已不能满足装备系统中信号处理的应用需求。具有大规模、高效数据处理能力的并行多DSP系统开始被广泛采用，越来越广泛地应用于雷达、声纳等测试仪和新装备的研制[1-2]。

并行DSP系统中节点间的数据传送是系统设计与实现中的重要问题。由于DSP硬件资源种类繁多，需要设计不同的软件满足硬件平台的异构和应用多样性，软件移植性差。因此需要一种构筑于软硬件间，为上层应用提供统一接口的通用软件系统，即中间件。中间件能屏蔽系统硬件差异，使开发的软件可复用、可移植，并可以对同一个硬件平台，实现不同应用。

并行系统节点间数据传递的通信模型主要分为消息传递和共享存储2类，其中，消息传递以其移植性和灵活性等优势被大多数并行系统采用。消息传递机制中应用较广泛的是消息传递接口(Message Passing Interface, MPI)，MPI移植性好、功能强大、效率高，是消息传递并行编程模式的标准。但由于DSP系统内存资源受限，应用MPI通信的开销较大，因此并行DSP系统无法满足应用MPI消耗的资源。目前MPI还没有一个真正面向DSP的产品，因此，设计与实现一种轻量级消息中间件是很必要的。

并行系统中的消息传递包括消息格式、交换机制、流量控制和路由选择等问题，其中，路由选择是实现消息可靠传递的关键。路径选择的高效性和路径信息的准确性直接影响系统性能和效率。

本文通过分析并行DSP系统资源受限和实时性要求等特点，基于Dijkstra算法，设计实现一种适合并行DSP系统应用的最短路径算法，用于通用声纳仿真测试仪中处理器间消息传递时查找路径。

2　并行DSP系统的路由特点与需求

路由通常分为静态路由和动态路由2种。静态路由根据系统网络拓扑信息事先设置路由表，在系统运行过程中路由表内容不变，用于系统拓扑结构相对固定的环境中，优点是高效、可靠。动态路由是网络中的路由器之间相互通信、传递路由信息、利用收到的路由信息更新路由表的过程，能实时地适应网络结构变化，适用于规模大、拓扑复杂的网络。

与通用并行系统相比，并行DSP系统有以下特点：(1)对于通用信号处理机(Commercial Off-The-Shelf, COTS)，硬件拓扑结构固定，可以支持多种数据流。(2)DSP片内存储资源有限，不支持动态存储管理。(3)DSP应用要求较高的实时性。因此，DSP系统采用静态路由，即在系统初始化时建立每个节点的转发表，存储最短路径信息，系统运行过程中转发表内容不变，节点转发表通过路由算法生成。

计算最短路径的算法有多种，其中，Dijkstra算法是目前许多工程解决最短路径问题时采用的理论基础，其主要思想是从源点求出长度最短的一条路径，通过对路径长度迭代得到从源点到其他各目标节点的最短路径。但不同实际问题有不同特点和不同应用条件，经典算法通常不能直接应用，必须针对具体情况加以修改[3-4]。

由并行DSP系统的特点和应用需求决定系统中的路由算法须满足以下条件：(1)查找时间短；(2)存储空间小。虽然采

作者简介： 王　哲(1981－)，女，硕士研究生，主研方向：接口技术；王希敏，副教授
收稿日期： 2009-03-15　**E-mail：** zhezhetou@yahoo.com.cn

用 Dijkstra 算法可以实现系统最短路径的查找，但该算法在执行时耗费大量空间存储和计算时间，因此，必须对 Dijkstra 算法加以改进，使之适用于并行 DSP 系统。

目前，实际应用中对 Dijkstra 算法的改进主要有 2 个方向：(1)路由信息存储方式的改进；(2)结合实际应用问题，增加算法限制条件，减少搜索次数。

本文采用存储空间相对小的邻接表存储并行DSP系统的硬件拓扑结构，将节点数据流信息作为路由算法搜索的限制条件，减少算法搜索次数和转发表的存储空间，降低算法复杂度，提高系统性能。

3 基于并行 DSP 系统的路由算法设计

算法的实现是将已知输入信息映射为需要的输出信息的过程。并行 DSP 系统将系统的硬件拓扑结构和节点数据流信息作为算法的输入条件，通过算法计算，将得到的路径信息输出作为系统节点的转发表。

3.1 并行 DSP 系统的硬件拓扑结构描述

假设已知某并行 DSP 系统的硬件拓扑结构，用 n 个节点的图 $G=(V,E)$描述，其中，$V=\{v_i|1\leqslant i\leqslant n, n$ 为系统节点数$\}$；$E=\{e_i|e_i=<v_i,v_j>, v_i, v_j\in V\}$；$\omega_1, \omega_2, \cdots, \omega_k$ 是图 G 的任意边上的权值，$\omega_i\geqslant 0(1\leqslant i\leqslant k)$；$\text{link}_i(1\leqslant i\leqslant m, m$ 是节点上的链路口数)是节点的链路口号。

采用邻接表存储 DSP 硬件拓扑结构，为每个节点 v 建立一个邻接表 node_port，邻接表的表元素用以下 3 项描述：节点 node、权值 weight 和链路口号 port，表示 v 的 1 条权值是 weight 的出边$<v$,node$>$，port 是 v 到达 node 的链路口号。DSP 系统硬件拓扑结构中每条边上的权值均相等，为了方便计算，令 weight=1。

采用标准 C++语言实现上述 DSP 硬件拓扑的描述，用顺序容器 list 存储拓扑图的邻接表。节点的邻接表的结构如下：

```
template<typename vertex> //模板类型定义
class node_port
{
vertex node; //vertex 是节点的类型
double weight;
int port;
    node_port(const vertex & x, const int & link, const
double & wet, const int& link)
 {
            node=x; weight=wet; port=link;
 }
};
```

DSP 硬件拓扑图中每个节点和与其关联的邻接表组成关联(节点，表)，n 个节点的系统有 n 个关联，用标准 C++的 STL 模板容器map存储此类关联，节点作为键，即map<vertex, list<node_port<vertex> > >。硬件拓扑图类包含图的属性和操作，拓扑图的类定义如下：

```
class graph
{
public:
    graph(const graph & g); //构造函数，用于构造一个空图
   bool contain_vertex(const vertex & v); //若图中含有顶点v，
//返回T，否则返回F
   void insert_vertex(const vertex & v); //若图中不含顶点v，
//则将v插入图中
    bool contains_edge(const vertex & v1, const vertex & v2);
    //若<v1,v2>是图的一条边，则返回T，否则返回F
    void insert_edge(const vertex & v1, const vertex & v2,
                 const double & wet, const int & link);
    //若边<v1,v2>不在图中，则插入这条边
    list<node_port<vertex> >get_incident_edges
                  (const vertex & v) const;
    //求与顶点v相关联的边，返回表(node, weight, link)
private:
    typedef map<vertex, list<node_port<vertex> > >Map;
    Map adj_map; //adj_map是Map类型的一个对象
};
```

3.2 DSP 系统节点数据流的描述

根据 DSP 系统节点的实际任务流向，提出系统节点数据流图[5]的设计。对于特定应用，DSP 系统节点有固定数据流向，用有向图 $G_r(V,E_r)$来描述节点间的数据流，V 是系统节点集合，E_r 是节点间数据流，箭头表示数据流向，由源节点指向目的节点。系统节点间的数据流如图 1 所示，该系统有 8 个节点，其中节点 1 向节点 0 和节点 4 输出数据；节点 2 只向节点 5 输出数据等。

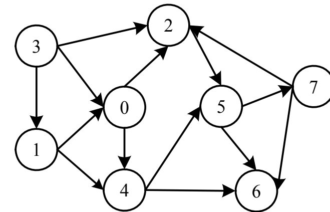

图 1 系统节点间的数据流

将与当前节点有数据流的节点称为该节点的一个任务节点。用集合容器 S_i 存储节点 i 的所有任务节点$(1\leqslant i\leqslant n)$，$S_i$ 称为节点 i 的任务节点集合，如图 3 中的节点 1，$S_1=\{0,4\}$。

每个节点对应一个任务节点集合，用关联(节点，任务节点集合)描述，用 map 容器存储此类关联，节点作为键，表示为 map<vertex, set<vertex>>。

3.3 节点转发表结构的描述

节点的转发表用于存储源节点到系统其他各节点的最短路径信息。在并行 DSP 系统中，节点间进行消息传递时，只要知道下一跳节点的相关信息，因此，只要将当前节点、目的节点、下一跳节点和相关链路口信息存入转发表即可。

用 list 链表存储节点转发表信息，转发表的结构如下：

```
struct PathInfo
{
    int currNode;  //本地节点
    int tagNode;  //目的节点
    int nextNode;  //从本地节点到目的节点所经过的第1个
//中间节点，若当前节点与目的节点是邻接节点，则令nextNode=
//tagNode
    int linkno;  //当前节点到nextNode的link口号
 };
```

3.4 路由算法的设计

针对经典 Dijkstra 算法消耗大量存储空间和计算时间的问题，在设计并行 DSP 系统路由算法时，将任务节点集合作为搜索的限制条件引入到算法实现中。主要改进思路如下：

(1)在算法开始时，对源节点 v 的任务节点集合 S_v 进行判断，如果为空，那么表明 v 不和系统中任何节点进行数据交互，直接结束算法，不必进行其他查找；如果不为空，那么继续执行。

(2)设 S 是已求得的各最短路径的终点集合，每次向 S 中插入一个节点 u，判断 u 是否在 S_v 中，若在，则将 u 从 S_v 中删除，直到 S_v 为空。此时，表明 v 的任务节点已在 S 中，立即结束算法，不必等到全部节点插入 S 中。

定义 并行 DSP 系统中路由算法的实现函数为 ShortPath_DSP()，使用优先队列 Q 选择最短路径。函数实现中其他变量含义如下：

w 是路径终点，dist[w]是 v 到 w 的路径长度；p_w 是从 v 到 w 的路径上 w 的直接前驱；link 是 p_w 到 w 通过的链路口号；映射 dist 存储各节点当前的最短路径长度，元素是(w,dist[v])；映射 pred 的元素是(w,p_w)；映射 linkNo 的元素是(w,link)；Q 的元素是(w,dist[w],link)。伪代码描述算法如下：

```
ShortPath_DSP(G, v, dist, pred, LinkNo, Sv)
输入 DSP系统硬件拓扑图G信息，包括节点、边、权值、节点的链路，v和Sv
输出 对于v的任务集Sv中任意一个节点w, dist[w], pw, link
  if v不在G中    return
  if Sv为空    rerurn
初始化
          S←v
          dist[v]←0;
          linkNo[v]←-1; //或其他不可能的值
          pred[v]←v;
          Q←(v,0,-1)
while pq不为空do
     取出Q中dist[u]最小的元素(u,dist[u])
     if u∈S continue
     else S.insert(u)
     if u∈Sv Sv.erase(u)
     if Sv为空break
     定义edge_list=G.get_incident_edges(u)//u的邻接表
     for iter←edge_list.begin() to edge_list.end() do
        定义w←iter_node
        if w不在S中do
        定义weight←iter_weight, port←iter_port
        if w不在dist映射中do
           dist[w]←d+weight
           linkNo[w]←port
           Q.push(w,dist[w],linkNo[w])
           pred[w]←u
      else if d+weight<dist[w]do
          dist[w]←d+weight
          linkNo[w]←port
          Q.push(w,dist[w],linkNo[w])
          pred[w]←u
```

4 基于 ADSPTS101 并行系统路由算法的实现

4.1 ADSPTS101 硬件结构

AD(Analog Devices)公司的 TigerSHARC 系列 DSP 芯片 TS101 是该公司继 SHARC 系列之后推出的一种新型高速实时数字信号处理芯片，是目前应用较为广泛的高端 DSP 并行处理芯片。簇式多处理器最高可支持 2 簇 8 片 ADSPTS101 的无缝连接，每片芯片具有 2 种类型的多处理器间接口：Link 口和 Cluster 总线。总线提供传输速率高达 800 MB/s 的多处理器间的数据通道，通过共享总线方式连接多处理器。每个 Link 口的传输速率高达 250 Mb/s，实现高效的点对点通信，支持共享总线存储与消息传递 2 种消息传递机制。

通用信号处理机采用 8 片 ADSPTS101 芯片作为处理器节点，ADSPTS101 并行系统硬件拓扑结构如图 2 所示。以该系统通过链路口方式通信为例，在 VisualDSP++运行环境中，给出上述路由算法的一个具体实例，验证算法的正确性。

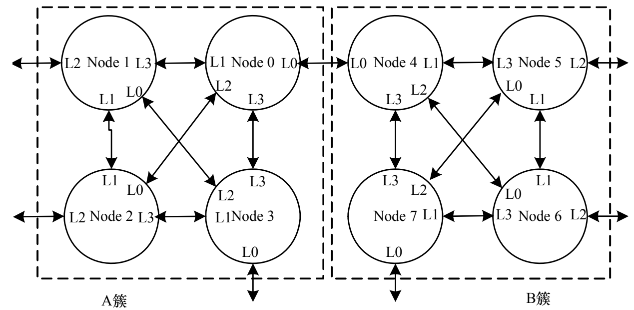

图 2 ADSPTS101 并行系统硬件拓扑结构

由上图可知，ADSPTS101 并行系统节点的邻接表信息，如，node0 的邻接表是(node1,1,link1)→(node2,1,link2)→(node3,1,link3)→(node4,1,link0)。系统初始化时，将此类信息通过图类操作 insert_vertex()和 insert_edge()赋给空图 g。

4.2 ADSPTS101 并行系统节点间的数据流

假设 ADSPTS101 并行系统节点的数据流如图 1 所示，用集合 S_i 表示第 i 个节点的任务节点集合 $0 \leqslant i \leqslant 7$，如 $S_0=\{2, 4\}$, $S_6=\{\varnothing\}$等。系统初始化时，通过集合容器的 insert 操作，将此类信息依次插入到各自任务节点集合中。

4.3 算法实现

调用最短路径函数 ShortPath_DSP()，得到指定源点到其任务节点的路径信息。例如，当前发送节点为 node7，node7 的任务节点集合为 $S_7=\{2,6\}$，利用上述设计的最短路径算法查找 node7 到其任务节点的最短路径。为测试算法的准确性，将运行结果输出，node7 到任务节点的最短路径如图 3 所示。

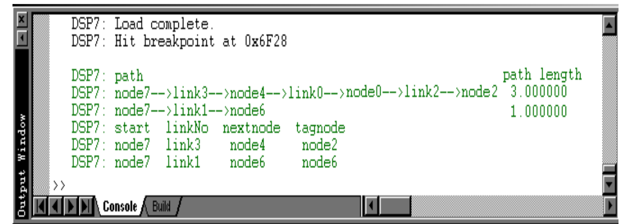

图 3 node7 到任务节点的最短路径

其中，path 部分是源节点到其任务节点的路径信息；path_length 是源节点到任务节点的最短路径长度，也是最小跳数。同时得到 node7 的转发表，信息表明，消息从 start 发送到 tagnode，经过 start 的 linkNo 口转发给 nextnode。

5 算法性能分析

算法的性能分析如下：

(1)空间复杂度分析。采用邻接表存储图的结构，空间复杂度是 $O(e)$，e 是硬件拓扑图的边数。尤其对于稀疏图，$e \ll n^2$，能大量节省存储空间。

(2)时间复杂度分析。算法的运行时间主要是对节点邻接表的搜索，搜索一条边的时间是 $O(\log_2 e)$。由于增加节点数据流信息作为算法搜索的限制条件，因此每次搜索只要考虑其任务节点，算法总时间小于 $O(\log_2 e)$，最坏的情况下为 $O(\log_2 e)$。

此外，由于任务节点数据流信息的增加，使原本存储所有节点路径信息的转发表只要存储任务节点的路径信息，节省了系统的内存资源，对于资源受限的 DSP 系统，有效提高了系统性能。 (下转第 246 页)