© 2001 Springer-Verlag. Reprinted with permission from G. Brebner and R. Woods (editors), Proceedings of the 11th Conference on Field-Programmable Logic and Applications (FPL 2001). Belfast, Northern Ireland, UK, 27-29 August 2001, pages 653-657.

Dijkstra's Shortest Path Routing Algorithm in Reconfigurable Hardware

Matti Tommiska and Jorma Skyttä

Signal Processing Laboratory
Helsinki University of Technology
Otakaari 5A
FIN-02150, Finland
{Matti.Tommiska, Jorma.Skytta}@hut.fi

Abstract. This paper discusses the suitability of reconfigurable computing architectures to different network routing methods. As an example of the speedup offered by reconfigurable logic, the implementation of Dijkstra's shortest path routing algorithm is presented and its performance is compared to a microprocessor-based solution.

1 Introduction

The uses of reconfigurable logic have increased both in number and scope. The combination of reconfigurability and high-speed computing has given birth to a new field of engineering: reconfigurable computing which ideally combines the flexibility of software with the speed of hardware. [1]

Increased Quality of Service (QoS) poses tough requirements on network routing. The increase in computational complexity is exponentially related to an increase in QoS, and to achieve acceptable network performance, additional computing resources are required [2]. A promising solution to computational bottlenecks in network routing is reconfigurable computing.

This paper presents a brief overview of the applications of reconfigurable computing in network routing. As a case study, an FPGA-based version of Dijkstra's shortest path algorithm is presented and the performance differences between the FPGA-based and a microprocessor-based versions of the same algorithm are compared.

2 Classification of Routing Methods and Suitability to Reconfigurability

There are a number of ways to classify routing algorithms [3,4]. One of the simplest routing strategies is static routing, where the path used by the sessions of each origin-destination pair is fixed regardless of traffic conditions.

Most major packet networks use adaptive routing, where the paths change occasionally in response to congestion. The routing algorithm should change its routes and guide traffic around the point of congestion.

G. Brebner and R. Woods (Eds.): FPL 2001, LNCS 2147, pp. 653–657, 2001.
© Springer-Verlag Berlin Heidelberg 2001

In centralized routing algorithms all route choices are made at the Routing Control Center (RCC). In distributed algorithms, the computation of routes is shared among the network nodes with information exchanged between them as necessary. Because computation is distributed evenly across the whole network, the network is not vulnerable to the breakdown of the RCC.

The applicability of reconfigurable computing in routing varies according to the routing method. In static routing, reconfigurable computing methods help in accelerating the computations to fill the lookup tables. In adaptive routing, reconfigurable computing allows the routing algorithm to run in hardware where parallelism is exploited to the fullest, and when network conditions change, a different routing algorithm is swapped in to run in the same hardware. Traditionally, adaptive routing algorithms have been run in software, but running these algorithms in reconfigurable hardware brings speed advantages.

Whether the routing method is centralized or distributed is not essential to the applicability of reconfigurable computing, since both central and distributed algorithms can be adaptive. The pros and cons of routing methods and the applicability of reconfigurable computing are presented in Table 1 [5].

Table 1. Characteristics of routing methods

Routing method	Advantages	Disadvantages	Applicability of reconfigurable computing
Static	Simple, fast	Inflexible	Precomputation of the routing tables
Adaptive	Adapts to network changes	Complex, requires careful planning	Good
Centralized	Relieves nodes from computation	Vulnerability of the RCC	Depends on the routing algorithm
Distributed	Large tolerance to link failures	Vulnerability to oscillations	Depends on the routing algorithm

3 Dijkstra's Shortest Path Algorithm in Route Computation

The problem of finding shortest paths plays a central role in the design and analysis of networks. Most routing problems can be solved as shortest path problems once an appropriate cost is assigned to each link, reflecting its available bandwidth, delay or bit error ratio, for example.

There are various algorithms for finding the shortest path if the edges in a network are characterized by a single non-negative additive metric. The most popular shortest path algorithm is Dijkstra's algorithm [6], which is used in Internet's Open Shortest Path First (OSPF) routing procedure [7].

Dijkstra's shortest path routing algorithm is presented below in pseudocode:

Given a network $G = (N, E)$, with a positive cost D_{ij} for all edges $(i, j \in N)$, start node S and a set P of permanently labeled nodes, the shortest path from start node S to every other node j is found as follows: Initially $P = \{S\}$, $D_S = 0$, and $D_j = d_{Sj}$ for $j \overset{j \neq S}{\in} N$.

Step 1: (Find the closest node.) Find $i \notin P$ such that

$$D_i = \min_{j \notin P} D_j$$

Set $P = P \cup \{i\}$. If P contains all nodes then stop; the algorithm is complete.

Step 2: (Updating of labels.) For all $j \notin P$ set

$$D_j = \min[D_j, D_i + d_{ij}]$$

Go to Step 1.

Since each step in Dijkstra's algorithm requires a number of operations proportional to |N|, and the steps are iterated $|\mathrm{N} - 1|$ times, the worst case computation is $\mathrm{O}(|\mathrm{N}|^2)$ [8]. Using priority queues the runtime of Dijkstra's algorithm is $\mathrm{O}(|\mathrm{E}| \lg |\mathrm{N}|^2)$, which is an improvement over $\mathrm{O}(|\mathrm{N}|^2)$ for sparse networks [9]. However, the space requirement increases and operations on priority queues are difficult to implement in reconfigurable logic, and for these reasons priority queues have not been dealt with in this paper.

4 FPGA-Based Dijkstra's Shortest Path Algorithm

A parameterizable version of Dijkstra's shortest path algorithm was designed in VHDL [10] with Synopsys' FPGA Express design software version 3.4 [11]. The design was targeted for Altera's FLEX10K device family [12] with the Quartus design software [13].

The parameterizable features of Dijkstra's shortest path algorithm were compiled into a separate VHDL package which was included in the main design file. This way the design of other versions of Dijsktra's algorithm with different accuracy and for networks of different sizes is made easier, since all the changes are made only in the VHDL package.

The block diagram of the FPGA-based Dijkstra's shortest path algorithm is presented in Fig. 1. The network structure is presented in the internal ROM block of the logic device. In the block diagram of Fig. 1, there are six address lines. This is sufficient to represent all node-to-node links of networks of size upto eight nodes, if the network is represented as an adjacency matrix [9]. The internal RAM blocks of the logic device represent the known status of nodes, the distance to this node and the previous node. There are separate address

and data lines for the ROM and RAM blocks, which allows the parallel transfer of information to/from the memory blocks. To speed up the computations for finding the closest node from the set of unknown nodes, all node-to-node links whose other pair is the last known node are prefetched from ROM. Then the next known node is computed by the parallel comparator bank.

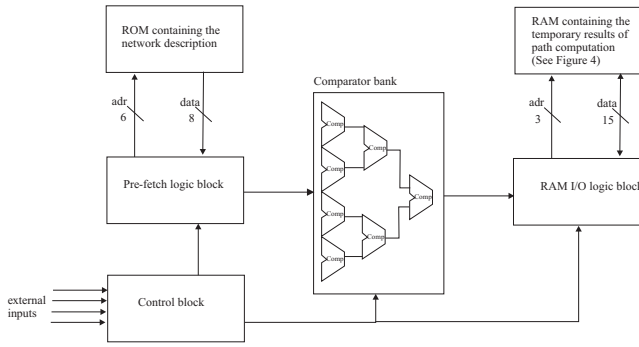

Fig. 1. The top-level block diagram of the FPGA-based Dijkstra's shortest path algorithm. The ROM block contains the network description and the RAM block contains the temporary results of shortest path computation. The comparator bank selects the smallest distance from the prefetched edge lengths.

As an example, the compiled version of Dijsktra's algorithm for networks of maximum sixe eight nodes fitted into an EPF10K20TC144-3 device, which has 1152 logic elements (LEs) corresponding to approximately 20000 available gates. The design required 72 per cent of all LEs and 5 per cent of available memory bits. Additional compilation results are summarized in Table 2. Logic element requirements increase linearly, since the size of the comparator bank grows linearly. On the other hand, memory requirements increase quadratically, since the network description of a network of size N nodes requires NxN memory locations. If the network description requires an external memory chip, all that is needed is to add an external data and address bus and to change the memory handling functions to handle external memory instead of internal memory.

To compare the performance of an FPGA-based Dijsktra's algorithm with a microprocessor-based version, an identical algorithm was coded in C and compiled with `gcc` in Linux Redhat 6.2. The same network descriptions were then tried in both the FPGA-based and software-based versions of the same algorithm. The speedup factor in favor of the FPGA-based version depended on the number of network nodes. As network sizes grew, the average execution time of the FPGA-based version grew only linearly, whereas the average execution time of the microprocessor-based version displayed quadratic growth (See Table 2). This can be attributed to the more effective FPGA-based execution of Step 1 in Dijkstra's algorithm, since multiple comparators are used in parallel (See Fig. 1).

Table 2. FPGA resources required by the implementation of Dijkstra's shortest path algorithm and a comparison between the execution times of FPGA-based and microprocessor-based versions of the same algorithm.

Nodes (edge cost @ 8 bits)	Logic Elements	Memory bits	Device	Execution time (FPGA-based)	Execution time (μP-based)	Average speedup factor
8	834	632	EPF10K20	10.6μs	250μs	23.58
16	1536	2116	EPF10K30	13.4μs	434μs	32.39
32	2744	8287	EPF10K50	17.2μs	802μs	46.63
64	5100	32894	EPF10K250	21.6μs	1456μs	67.41

5 Conclusions

Reconfigurable architectures have many applications in network routing. Depending on the routing algorithm or method, reconfigurability may assist in speeding up network routing.

The FPGA-based version of Dijkstra's shortest path algorithm was tens of times faster than a microprocessor-based version. This can be attributed to the following factors: multiple assignments to variables are executed concurrently, multiple arithmetic operations, including comparisons, are executed in parallel and the data structures and tables are implemented in the internal memory blocks.

References

1. N. Tredennick: "Technology and Business: Forces Driving Microprocessor Evolution", Proceedings of the IEEE, Vol. 83, No. 12, December 1995, pp. 1641-1652.
2. A. Alles: ATM Internetworking. Cisco Systems, Inc. 1995.
3. W. Stallings: Data and Computer Communications. Prentice-Hall, 1999.
4. A. Tanenbaum: Computer Networks. Prentice-Hall, 1996.
5. M. Tommiska: "Reconfigurable Computing in Communications Systems", Licentiate of Sciences Thesis, Helsinki University of Technology, 1998, pp. 46-50.
6. E. Dijkstra: "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, Vol. 1, 1959, pp. 269-271.
7. J. Moy: "OSPF Version 2, RFC 2328", May 1998.
8. D. Bertsekas, R. Gallager: Data Networks. Prentice-Hall, 1987, pp. 297-421
9. M.A. Weiss: Data Structures and Algorithm Analysis in C. The Benjamin/Cummings Publishing Company, Inc. 1993, pp. 281-343
10. M. Zwolinks: "Digital System Design and VHDL", Prentice-Hall 2000.
11. FPGA Express User's Manual, 2000, Synopsys Corporation.
12. FLEX 10K Embedded Programmable Logic Family Data Sheet, ver. 4.02, May 2000, Altera Corporation.
13. Quartus Programmable Logic Development System & Software, ver. 1.01, May 1999, Altera Corporation.