Proc. IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Berlin, vol. 1, pp. 339-344, 1997

A Simple Hardware Solution for Fast Computation of Shortest Paths

R. Möller
University of Tübingen, Dept. Computer Architecture
Köstlinstr. 6, D-72074 Tübingen, Germany
e-mail: moeller@informatik.uni-tuebingen.de

P. Paschke
Technical University of Ilmenau, Dept. Neuroinformatics
Gustav-Kirchhoff-Str. 2, D-98684 Ilmenau, Germany
e-mail: Peter.Paschke@Informatik.TU-Ilmenau.DE

Keywords: shortest paths, parallel hardware, graph-searching

ABSTRACT

The computation of shortest paths can be accomplished by graph-searching algorithms. These algorithms are based on a time-consuming computation of potential vectors. A new approach to parallelization is presented, which enormously reduces the complexity of an implementation in a digital hardware. It is based on the transformation of costs into time-delays.

INTRODUCTION

The computation of shortest paths in maps of the environment is a basic task of robotics. The two main approaches to this problem are graph-searching algorithms [1, 7] and artificial potential fields [5, 6]. Artificial potential fields suffer from the existence of undesired local minima, whereas graph-searching algorithms do not have that problem but are often considered time-consuming [3]. Attempts have been made to accelerate graph-searching algorithms using a fine-grained, highly parallel hardware with locally interconnected nodes [4]; they are based on a direct mapping of the algorithm to the hardware. This paper presents another approach to a parallel hardware that starts from a reformulation of the graph-searching algorithms where distances are transformed into time. With this approach, the hardware effort and the time for computation are enormously reduced.

BASIC GRAPH-SEARCHING ALGORITHMS

Graph Model

The environment of the robot is encoded as a graph: nodes represent positions in space and edges are assigned weights w_{ij} encoding the costs for the transition from node i to node j. In the following, costs will be restricted to geometrical distances. The algorithms discussed below are 'all-shortest-path' algorithms, computing the paths with minimal sum of costs from all nodes to a reference node. In the first run, a potential vector $\underline{p} = (p_1, \ldots, p_n)$ is computed, where p_i is the shortest distance from node i to the reference node. The potential vector is used for path tracking in the second run.

Computation of Potential Vectors

Many 'all-shortest-path' algorithms are variations of DIJKSTRAs algorithm [1]: Initially, all nodes are uncoloured; the potentials are $p_k = +\infty \ \forall k, k \neq z$ and $p_z = 0$, with z being the reference node.

Initially, index i is set to z.

1. Colour node i.
2. For each uncoloured node j receiving an edge from i, recompute its potential by

$$p_j := \min\{p_j, p_i + w_{ij}\} \tag{1}$$

3. From the uncoloured nodes with finite potential select the node i with minimal potential.
4. If there is no such node, stop, otherwise proceed with 1.

For the comparision with the performance of the hardware an improved version of the above method is used. It benefits from a discretization of the weights: the minimum search in step 3 is avoided by a partition into sets of nodes with equal potential. In addition, step 4 is modified, so that the computation is finished at the node representing the initial position. Both the hardware and this improved algorithm compute identical potential vectors, shown in figure 1 (center) for an example from mobile robotics with a grid-like graph (left).

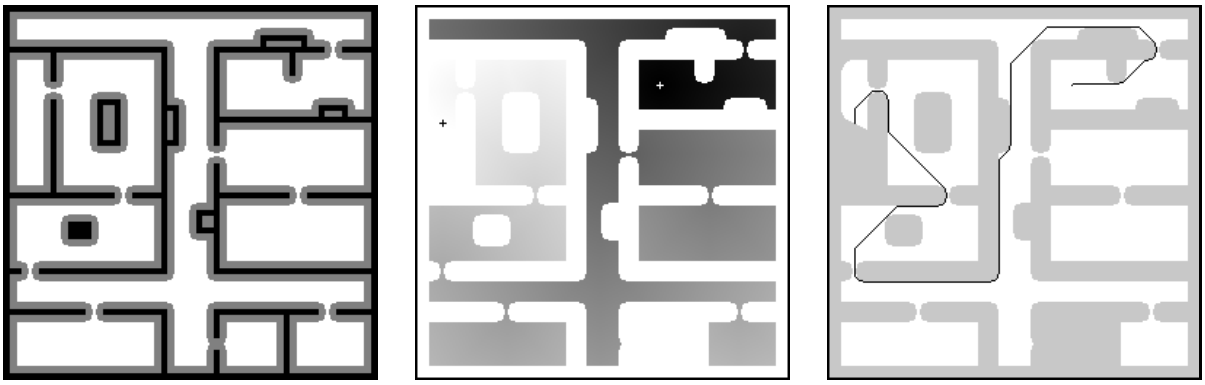

Figure 1: *Left: Example map (256×256 nodes). Black: obstacles of the original map, gray: map dilated by roboter size. Center: Potential map (termination at stop node). Gray values encode potentials, black being* 0*. Black cross: stop node, white cross: reference node. Right: Gray: nodes with output* $Y = 0$ *when the stop node was reached. Black: shortest path calculated by sequential backtracking (method by* DREYFUS*).*

Path Tracking

A sequential tracking procedure like the one proposed by DREYFUS [2] applied to the potential vector provides a shortest path from the initial to the target position: Starting at the node i encoding the initial or current position, the next node on the path is a node j sending an edge to i, for which

$$p_i = p_j + w_{ji} \tag{2}$$

is valid. This is repeated until the reference node is reached. For most applications it will be sufficient to compute only the *local* course; in this case, only one or a few steps with equation (2) are executed. For the example map, the resulting path from a complete tracking is drawn in figure 1 (right).

PRESENT APPROACH TO PARALLEL HARDWARE

Algorithm

In [4], another variation of DIJKSTRAs algorithm is presented, which is suited for an implementation of the potential computation (being the time-consuming part of the path-planning) in a fine-grained parallel hardware. Whereas in equation (1) the potential of a node i is *distributed* to all nodes receiving edges from i, in this version each node i *gathers* the potentials p_j of all nodes sending edges to i ($j \in \Pi_i$), adds the corresponding weights w_{ji}, and updates its own potential p_j, if the result is below p_i:

$$p_i := \min_{j \in \Pi_i} \{p_i, p_j + w_{ji}\}. \tag{3}$$

This computation can be performed for all nodes in parallel, until the potentials converge.

Hardware

For grid-like graphs, where a node is connected to its immediate 8 neighbours, [4] describes a hardware solution that is an immediate projection of equation (3) into a parallel hardware: each graph-node is assigned a hardware node performing this computation. Therefore, a full parallel computation requires 8 comparators and 8 adders for each node. Furthermore, potentials have to be exchanged between the nodes via parallel lines. This high effort conflicts with the low inherent parallelism of the problem: only the nodes at a wave front spreading from the reference node update their potential, each node usually only once or a few times.

NEW APPROACH TO PARALLEL HARDWARE

Algorithm

The approach to parallelization presented here starts from a reformulation of the algorithm: instead of summing up weights using adders, the computation is based on the summation of time-delays proportional to the distances between the nodes. This is much easier to realize in a digital hardware. Starting at the reference node z, a binary wave spreads across the nodes, until the stop node is reached. Between the nodes only a single bit is passed with a delay proportional to the distance. When the wave arrives at a node, the node registers the time elapsed since the wave has been started; this time corresponds to the potential of that node. It is another advantage compared to the algorithm in [4], that the computation can be terminated as soon as a stop node is reached.

Hardware

In a digital hardware, as shown in figure 2, the measurement of time can be realized by a global counter (Cntr.) and latches (Reg.) in each node. The global counter starts from 0 and is incremented synchronously with the delay elements. As soon as the binary wave arrives at a node, the latch of that node is locked and preserves the current value of the global counter which corresponds to the potential of the node. Nodes that have not been reached when the wave arrived at the stop node should be assigned ‘infinite’ potential. This can be achieved in a simple way if the computation is continued for one step. The value of the global counter can then be interpreted as infinity.

The hardware benefits from simplifications for the special field of shortest-path problems, namely regular, grid-like graphs and symmetric weights with only three values (between a node and its 4

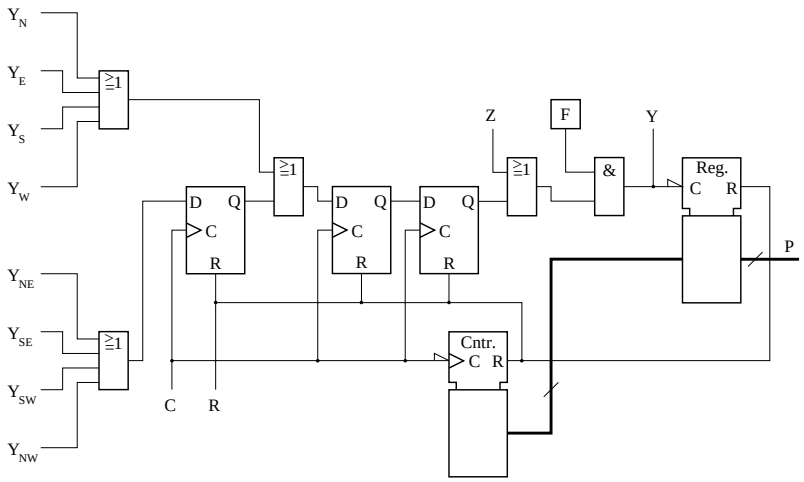

Figure 2: *Circuit of a single node, and global counter (Cntr). $Y_N \ldots Y_{NW}$ denote the outputs of neighbouring nodes. F is a 1-bit memory deciding between free space (1) or obstacle (0). At C, the clock signal is fed in, at R the reset signal. Z = 1 identifies a reference node. The output signal of a node is marked by Y. After the latch (Reg) is locked due to Y = 1, P gives the potential of the node.*

orthogonal resp. 4 diagonal neighbours, or infinity, if one of two connected nodes is an obstacle). In a digital hardware, the signal delays have to be integer. The distances 1 and $\sqrt{2}$ to the 8 immediate neighbours are approximated by time-delays of 2 and 3 cycles in the hardware of figure 2[1]. These delays can be integrated into a single delay line with only 3 D-flip-flops. It is fed by a combined signal of the outputs ($Y_N \ldots Y_{NW}$) of the orthogonal as well as the diagonal neighbours. The end of the delay line is connected to the lock input of the latch.

To encode infinite weights between nodes, it is sufficient to prevent an obstacle node from being stopped and from emitting $Y = 1$. For this purpose, between delay line and latch an AND-gate and a one-bit memory F is inserted that suppresses the transmission of the signal 1, if $F = 0$, i.e., if the node does not belong to the free space. In a similar way the reference node is encoded: a one-bit memory or signal line Z and an OR-gate will force an output of $Y = 1$ if $Z = 1$ and lock the latch — the reference node is clamped to potential 0.

Compared to the hardware described in [4], the complexity of a node is radically reduced: instead of 8 adders and 8 comparators, there are only 3 delay elements, a latch and some logic gates. Furthermore, the signal transmission between the nodes is reduced to a single bit (Y).

SIMULATIONS

Shortest Paths

The hardware of figure 2 has been tested in simulations; all examples presented here relate to a map of the size 256×256 nodes. The resulting potential vector is shown in figure 1 (center); this vector is identical to the vector obtained from the improved DIJKSTRA-algorithm. As visible in figure 1 (white areas in the middle picture, gray areas in the right picture), the computation was terminated at the stop node, so some of the free-space nodes in reach are not stopped and therefore carry 'infinite' potential. The path length and, therefore, the potential of the stop node are 1355 cycles, the computation was stopped at cycle 1356. With only 20 MHz assumed as the basic clock, the computation would be

[1]This should be a sufficient approximation: the discretization error is small compared to the error resulting from the approximation of the wave-front as an octogon when the 8 immediate neighbours are used.

completed within 68 μs. The improved DIJKSTRA-algorithm coded in C and executed on a 133 MHz Pentium PC takes approx. 63 ms, resulting in a speedup of approx. 1000.

Safe Paths

Another possible application of the hardware is the computation of 'safe' paths running in safe distance from obstacles. For that purpose, the Z input in figure 2 has to be realized as a one-bit memory. All nodes are assigned $F = 1$ to mark them as part of the free space, and all obstacle nodes have $Z = 1$. The potentials p_i^d obtained from this computation can be interpreted as the distance to the nearest obstacle; see figure 3 (left). The computation can be terminated at a cycle corresponding to the maximal distance in an empty map (length of the diagonal line), which is 768 resp. 38 μs with 20 MHz for the example map.

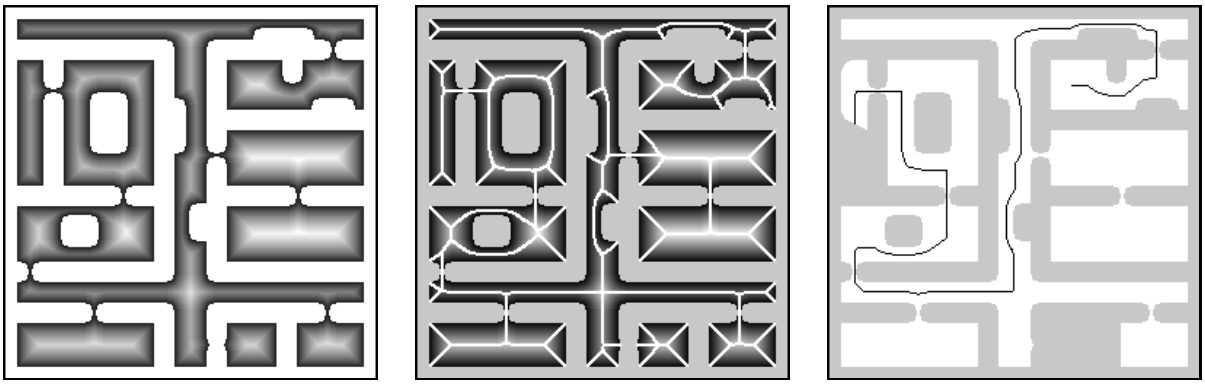

Figure 3: *Left: Potential vector $\underline{p}^d$ obtained for reference points at all obstacles, potential 0 is black. Center: Skeleton computed with a local algorithm from potential vector $\underline{p}^d$. Right: Safe path calculated from $\underline{p}^t$ and $\underline{p}^d$.*

There are several possible ways to incorporate both this potential vector $\underline{p}^d$ and the vector $\underline{p}^t$ obtained for the reference node z at the target position in a tracking procedure for safe paths. Fast local methods, that can be restricted to the first few steps of a path without the need to compute the complete path, should be preferred, because deviations from the path as well as the availability of new sensor data will require a recomputation anyway. The simple method used here searches for a node with maximal distance to an obstacle among all neighbouring nodes, that are nearer to the reference node z than the current path node i:

$$\hat{p}_i^d = \max_{j \in \Pi_i'} \{p_j^d\}, \quad \Pi_i' = \{k \in \Pi_i : p_k^t < p_i^t\} \tag{4}$$

Then, in a second pass, among the neighbours having the maximal obstacle distance $\hat{p}_i^d$, the neighbour with minimal distance to the reference node is identified; this node becomes the next path node i':

$$i' = \arg\min_{j \in \Pi_i''} \{p_j^t\}, \quad \Pi_i'' = \{k \in \Pi_i' : p_k^d = \hat{p}_i^d\} \tag{5}$$

The resulting path is shown in figure 3 (right). In contrast to artificial potential fields (that also consider the distance to obstacles) it is guaranteed, that a safe path to the reference node can be found, as long as the reference node is reachable from the initial node.

Another possible application of the method is to compute a kind of 'skeleton' of the map (similar to the generalized Voronoi diagram [6, 8]); see figure 3 (center). This can be achieved by a purely local procedure (involving only the 5×5 neighbourhood of a node) that detects all points at the 'crest' of the potential vector $\underline{p}^d$.

CONCLUSION

The hardware presented could be shown to be suited for the very fast computation of shortest as well as of 'safe' paths. It should be especially advantageous for the computation of paths in large, high-resolution maps, where paths have to be recomputed along the acquisition of new data for the model, if the target point changes (pursuit [3]), or for complex planning algorithms where different, alternative paths are requested.

Compared to the proposed hardware in [4], the hardware effort is radically reduced, enabling the VLSI implementation of large grids on a single chip. The circuit shows basically the same construction as a RAM memory and could be implemented and used in a similar way.

The delay line of the nodes could be prolonged by additional flip-flops and expanded with multiple, switchable entry points into the line to encode different weights. This would extend the possible application of the hardware from shortest-path problems in the strict geometrical sense to more general minimal-cost problems, e.g. including special penalties. Furthermore, an expansion to higher-dimensional maps is possible.

REFERENCES

[1] E.W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[2] S.E. Dreyfus. An appraisal of some short-path algorithms. *Operations Research*, 17:395–412, 1969.

[3] R. Glasius, A. Komoda, and S. Gielen. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1):125–133, 1995.

[4] Mohamad H. Hassoun and Ashvin J. Sanghvi. Fast Computation of Optimal Paths in Two- and Higher-Dimensional Maps. *Neural Networks*, 3:355–363, 1990.

[5] Oussama Khatib. Real-time obstacle avoidance for manipulators and robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[6] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, Dortrecht, London, 1991.

[7] E. Minieka. *Optimization Algorithms for Networks and Graphs*. Dekker, New York, Basel, 1978.

[8] Sebastian Thrun and Arno Bücken. Learning maps for indoor mobile robot navigation. Technical Report CMU-CS-96-121, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1996.