Architectural Design of Neural Network Hardware for Job Shop Scheduling

P.B. Luh, X. Zhao, L. S. Thakur
University of Connecticut, Storrs, CT 06269-2157, USA
K. H. Chen, T. D. Chiueh, S. C. Chang
National Taiwan University, Taiwan
Submitted by J. M. Shyu (1),
National Space Program Office, Hsin-Chu City, Taiwan

Abstract

By combining neural network optimization ideas with "Lagrangian relaxation" for constraint handling, a novel Lagrangian relaxation neural network (LRNN) has recently been developed for job shop scheduling. This paper is to explore architectural design issues for the hardware implementation of such neural networks. A digital circuitry with a micro-controller and an optimization chip is designed, where a parallel architecture and a pipeline architecture are explored for the optimization chip. Simulation results show that the LRNN hardware will provide near-optimal solutions for practical job shop scheduling problems. It is estimated that the parallel architecture will obtain one order of magnitude speed gain, and the pipeline architecture will obtain two orders speed gain as compared with the currently used method.

Keywords: Job shop scheduling, neural network, hardware design

1 INTRODUCTION

Manufacturing scheduling is an important but difficult task, and is often formulated as a combinatorial optimization problem [1,4]. To effectively solve this class of problems, a novel optimization method has recently been developed by combining neural network optimization ideas with "Lagrangian relaxation" for constraint handling. This paper is to explore architectural design issues for the hardware implementation of such neural networks for job shop scheduling.

Historically, Hopfield-type neural networks were developed for unconstrained optimization based on the "Lyapunov stability theory" of dynamic systems: if a network is "stable," its "energy" will decrease to a minimum as the system approaches its "equilibrium state." If one can properly set up a network that maps the objective function of an optimization problem onto an "energy function," then the solution is a natural result of network convergence, and can be obtained at a very fast speed [3].

For constrained optimization, the Hopfield-type networks convert a constrained problem to an unconstrained one by having penalty terms on constraint violations [3]. A tradeoff between solution optimality and constraint satisfaction has to be made through the fine tuning of penalty coefficients. The tradeoff, however, is generally difficult to make. In addition, Hopfield-type networks may possess many local minima. Since escaping from local minima is not an easy task [9], the solution quality depends highly on initial conditions.

Hopfield-type networks and its variations have been developed for job shop scheduling [2, 10]. Although these models demonstrate the possibility of using neural networks for solving small scheduling problems, they suffer from the above-mentioned difficulties. It is not easy to scale up these methods to solve practical problems. Heuristics have also been used to modify neuron dynamics to induce constraint satisfaction within the job shop context [8]. The results, however, may be far from optimal.

Recently we have developed a novel Lagrangian relaxation neural network (LRNN) method by combining neural network optimization ideas with "Lagrangian relaxation" for constraint handling [5]. This paper is to explore architectural design issues for the hardware implementation of such networks for job shop scheduling. Our goal is to implement LRNN in hardware to drastically reduce the computation time for practical size problems, and to provide an on-line near-optimal scheduling system. In Section 2, LRNN for job shop scheduling is presented. In section 3, a digital circuitry with a micro-controller and an optimization chip is designed to implement LRNN, were a parallel architecture and a pipeline architecture are explored for the optimization chip. Simulation results in Section 4 show that the LRNN hardware will provide near-optimal solutions for practical job shop scheduling problems. It is estimated that the parallel architecture will obtain one order of magnitude speed gain, and the pipeline architecture will obtain two orders speed gain as compared with the currently used method.

2 LRNN FOR JOB SHOP SCHEDULING

2.1 Problem Formulation

In a job shop, there are *H* machine types, and each machine type may consist of a few identical machines [6]. There are *I* parts to be scheduled over a horizon of *K* time units, and part *i* has its due date D_i, weight (or priority) W_i, and requires J_i sequential operations for its completion. Each operation is to be performed on a machine of a specified type for a period of time, satisfying the *processing time requirements*. The processing may start only after its preceding operation has been completed, satisfying the o*peration precedence constraints*. Furthermore, the number of operations assigned to machine type *h* at time *k* should be less than or equal to M_{kh}, the number of machines available at that time, satisfying the *machine capacity constraints*:

$\sum_{ij} \delta_{ijkh} \le M_{kh}$, $k = 1, ..., K$; $h \in H$. (1)

Here δ_{ijkh} is a 0-1 "operation-level" variable. It equals 1 if operation j of part i is being performed by machine type h at time k, and 0 otherwise.

The time-based competition goal of on-time delivery is modeled as a penalty on delivery tardiness $T_i = \max [0, C_i - D_i]$, where C_i is the completion time for part i. The objective function is the total weighted part tardiness $\sum_{i=0}^{I-1} W_i T_i^2$, and the problem is to determine operation beginning times b_{ij} (or completion time c_{ij}) for individual operations to minimize the objective function. With linear constraints and additive objective function, the key feature of this formulation is its *separability* [6].

2.2 Solution Methodology

We have applied Lagrangian relaxation to the above problems, and developed an efficient near-optimization method for job shop scheduling [6]. Within the LR framework, machine capacity constraints are relaxed by using Lagrange multipliers π_{kh}, and the "relaxed problem" is given by

$$\min_{\{b_{ij}\}} L, \text{ with } L \equiv \sum_i W_i T_i^2 + \sum_{ij} \sum_{k=b_{ji}}^{c_{ij}} \pi_{kh} - \sum_{kh} \pi_{kh} M_{kh}, \quad (2)$$

subject to individual part constraints. Since the formulation is *separable*, the relaxed problem can be decomposed into the following decoupled *part subproblems* for a given set of multipliers:

$$\min_{\{b_{ij}\}} L_i, \text{ with } L_i \equiv W_i T_i^2 + \sum_{j=1}^{J_i} \sum_{k=b_{ij}}^{c_{ij}} \pi_{kh}, \; i = 1, \ldots, I. \quad (3)$$

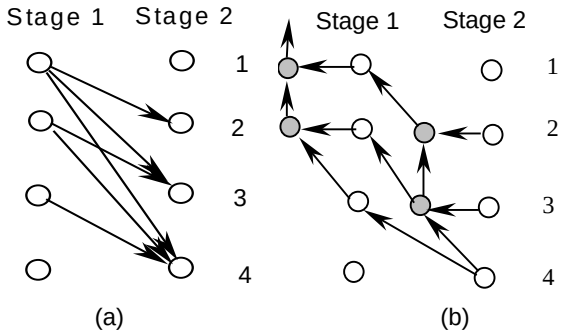

Figure 1: DP and NBDP.

Each part subproblem can be efficiently solved by using dynamic programming (DP) [6], with its structure shown in Figure 1 (a). With stages corresponding to operations and states corresponding to operation beginning times, the backward DP algorithm starts with the last stage, and computes the tardiness penalties and machine utilization costs. The stage-wise machine utilization cost is the summation of P_{ij} related multipliers, where P_{ij} is the operation processing time. As the algorithm moves backwards, cumulative costs are computed based on the stage-wise costs and the minimum of the cumulative costs for the succeeding stage, subject to allowable state transitions as delineated by operation precedence constraints. This minimization can be efficiently implemented by pair-wise comparisons, starting from the last state of the succeeding stages. The optimal subproblem cost is then obtained as the minimum of the cumulative costs at the first stage, and the optimal beginning times for individual operations can be obtained by the "forward sweep" which traces the optimal beginning times forward in stages.

Let L_i^* denote the minimal subproblem cost of part i with given multipliers, the high level Lagrangian dual problem is then obtained as follows:

$$\max_{\{\pi_{kh}\}} D, \text{ with } D \equiv \sum_i L_i^* - \sum_{k,h} \pi_{kh} M_{kh}. \quad (4)$$

It can be shown that the dual function is always concave, and provides a lower bound to the original problem. With the minimum subproblem solutions for given multipliers π_{kh}, the subgradient g of the dual function D is calculated by

$$g_{kh} = \sum_i \sum_j \delta_{ijkh} - M_{kh}. \quad (5)$$

Iterative adjustment of multipliers along the subgradient directions with proper step sizes, repeated resolution of subproblems, and the final heuristic adjustment of subproblem solutions lead to a near-optimal feasible solution of the original problem. The quality of the schedule can be quantitatively evaluated by the duality gap, which is the relative difference between the feasible cost and the maximum dual cost [6].

2.3 Lagrangian Relaxation Neural Networks

Lagrangian relaxation has recently been combined with neural networks to solve constrained optimization problems [5]. Since the dual function is always concave, the key idea of LRNN is to create a network to let the negative dual be the energy function. If this can be done, then the negative dual will naturally approach its minimum (or the dual will approach its maximum) as the network converges. However, since the negative dual is not explicitly available but must be obtained through the resolution of the relaxed problem for various multipliers, the construction of the network is a bit complicated. The crux of LRNN is to merge these two constructs, one for the negative dual and the other for the relaxed problem, and let them feed each other and converge simultaneously. In LRNN, the network elements that update multipliers will be referred to as the *"Lagrangian neurons."* In contrast, neurons minimizing the subproblems will be called *"decision neurons."* At any instant of time, however, decision neurons may not provide minimum solutions of the relaxed subproblems for the current π_{kh}. In spite of this, a direction is calculated according to (5) based on these subproblem solutions, and Lagrangian neurons π_{kh} are updated along the "surrogate subgradient direction" obtained.

Neuron-based dynamic programming (NBDP) is developed to solve part subproblems, and can effectively handle the local constraints as well as the integer variables involved [5]. The key idea is to make the best use of the DP structure that already exists, and implement the DP functions by neurons. In doing this, the DP structure illustrated in Figure 1 (a) is utilized, where each state is represented by a neuron to obtain the cumulative cost by adding up two values: the stage-wise cost derived from multipliers and the minimum cumulative costs of the succeeding stage. The *pair-wise comparison* to obtain the minimum cumulative costs of the succeeding stage is carried out through the introduction of another layer of "*comparison neurons*." The connections of comparison neurons and "state neurons" are subject to state transitions as shown in Figure 1 (b), where comparison neurons are represented by gray circles. The traditional backward DP algorithm is thus

mapped onto a neural network with simple topology and elementary functional requirements that can be implemented in hardware.

The structure of LRNN consists of Lagrangian neuron updating and NBDP for subproblem solving as shown in Figure 2 (only one NBDP is shown here). Within each NBDP, signals propagate from the last stage to the first stage, and forward sweep is then used to obtain subproblem solutions. Unlike the traditional LR method, multiplier updating in LRNN does not wait for all subproblems to be solved. Instead, updating directions are calculated after only one of the subproblems is solved, and Lagrangian neurons are updated accordingly. The convergence of LRNN has been proved in [5].

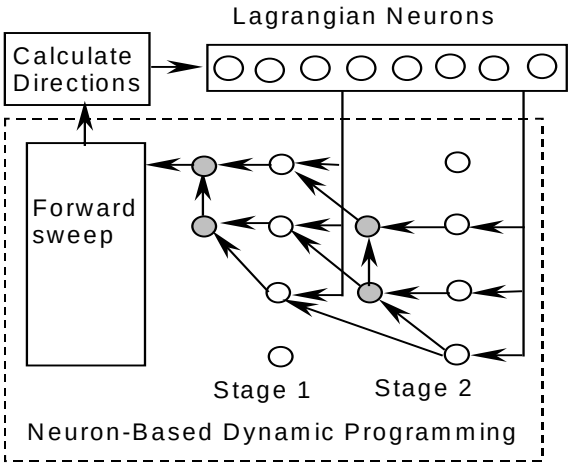

Figure 2: Structure of LRNN.

3 ARCHITECTURAL DESIGN

3.1 Hardware Architecture

A digital circuitry will be used to implement the above LRNN to be embedded within a personal computer. In hardware design, key considerations include speed, hardware complexity, accuracy, chip area, I/O requirements, and the flexibility for different problem sizes [7]. By trading off hardware complexity vs. numerical accuracy, 16-bit integer instead of floating point calculation is used. Since general multiplication consumes chip area, only multiplication by power of two is considered and implemented by simple bit shifting. In view that NBDP is a common module for solving individual subproblems and is quite complicate to implement, a chip contains one NBDP to allow flexibility regarding the number of parts to be scheduled. Furthermore, multiplier updating is built within this chip to avoid large amount of I/Os that would be required otherwise. The resulting hardware architecture consists of two components: a micro-controller and an optimization chip as shown in Figure 3.

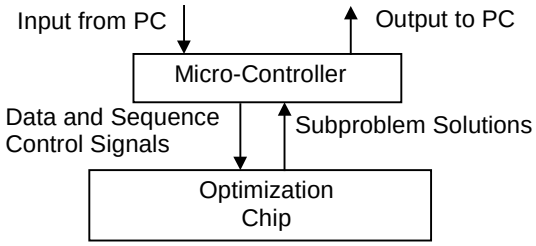

Figure 3: Digital hardware architecture.

The micro-controller feeds job shop input data from PC to the optimization chip, controls its processing sequence, and returns its solutions to PC. The optimization chip is used to perform NBDP and multiplier updating. After completing NBDP for one subproblem, multiplier updating directions are calculated based on the new subproblem solution to update the multipliers. Another subproblem is then loaded. Since the optimization chip is the key part of the hardware, a parallel architecture and a pipeline architecture are designed as presented next.

3.2 Parallel Architecture

In the parallel architecture, the calculations for one stage of NBDP as illustrated in Figure 2 are implemented by K "state cells" in parallel as shown in Figure 4. State cell k has one adder and one comparator to implement the "state neuron" and the "comparison neuron" for state k. It also stores multipliers and directions for all machine types at time k in its local memory, and updates these directions and multipliers. All state cells calculate their stage-wise costs and cumulative costs in parallel, and this takes $P_{ij}+1$ clock cycles. The pair-wise comparison is then performed sequentially from the last cell to the first cell, and a 0-1 "minimum-indicating" bit is stored in each cell's local memory to be used in the forward sweep (J bits are required for all stages). This sequential comparison takes k clock cycles. Therefore, one-stage NBDP will take a total of $P_{ij}+K+1$ clock cycles. Under the control of an internal sequence controller, various stages of NBDP for a subproblem are sequentially performed from the last stage to the first. A simple circuit then carries out the forward sweep function by tracing through the minimum-indicating bits from the first stage to the last. This takes αKxJ clock cycles, where α can be easily made smaller than 0.05 by a specific "tracing circuit." With the new subproblem solution available, directions are adjusted and multipliers are updated by state cells in parallel. This process is controlled by the sequence controller and requires only a few clock cycles. The total time required to complete one subproblem is approximately $K \times J$ clock cycles, with the sequential pair-wise comparison as the bottleneck.

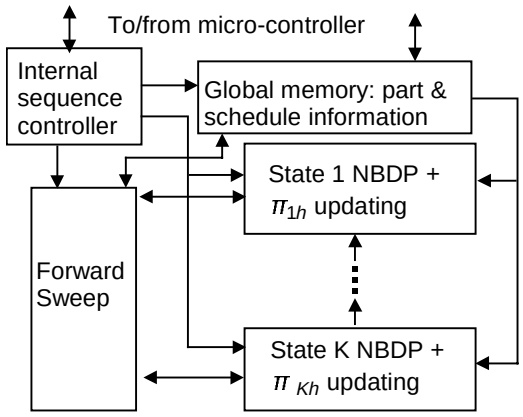

Figure 4: Parallel architecture with K state cells.

3.3 Pipeline Architecture

The heart of the pipeline architecture is J "stage cells" each with three adders and one comparator as depicted in Figure 5. The calculation for stage cell j starts from the bottom state with the largest beginning time. For state k of stage cell j, two adders (SC) are first used to obtain the stage-wise cost based on the stage-wise cost from state $k+1$. One adder (CC) then gets the cumulative cost using the corresponding minimal cost from stage $j+1$, and the comparator (MC) finally finds the minimum. As shown in Figure 5, the calculations for neighboring states within one stage are properly pipelined. The processing of a subproblem starting from stage cell J to stage cell 1 is also properly pipelined. In this way, the calculations for

all stages can be finished within approximately K clock cycles. The forward sweep is then similarly implemented as that of the parallel architecture except that the minimum-indicating bits are stored in the forward sweep circuit. Since every stage cell needs multipliers of all time indices, global memory is needed to store all multipliers. This implies that a "multi-reading" capability has to be designed to facilitate concurrent access of multipliers by different cells. A multiplier updating circuit separated from the stage cells is also required to update multipliers in parallel. The total time required for solving one subproblem is approximately K clock cycles as compared to $K \times J$ for the parallel architecture.

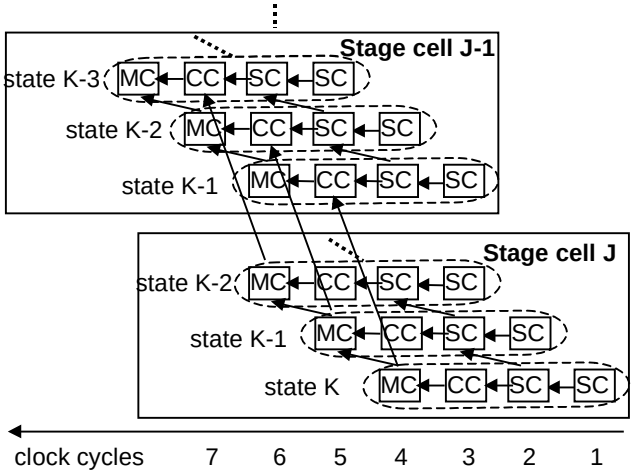

Figure 5: Pipeline architecture with J stage cells.

4 SIMULATION AND PERFORMANCE ANALYSIS

The above hardware design has been simulated on a Pentium II 400 MHz PC. The simulation is to evaluate the performance of LRNN under 16-bit integer calculation and fixed step size (2^{-n}) multiplier updating. Initial testing showed that T_i^2 in the objective function frequently caused overflows. Thus T_i is used instead of T_i^2 in the following simulations. Results of three problems after 600 iterations are presented in Table 1. The small duality gaps imply that near-optimal solutions are obtained, and the solution quality of LRNN is similar to those obtained by a currently used LR method [6].

Data (I/J/K/H)	Dual	Feasible	Gap
20/20/ 500/11	268	302	13%
40/20/1000/7	522	544	4%
80/20/2000/8	2650	2918	10%

Table 1: Simulation results for LRNN.

To compare the speed of LRNN with that of LR, a problem with *I/J/K/H* = 500/20/5000/10 is considered. This problem is first solved by LR, and it takes one hour to run 600 iterations. Under the assumption of 100 MHz operating frequency for hardware implementation, the time required for the two LRNN architectures are estimated in Table 2. The parallel architecture will cut the computation time to be within 5 minutes, and the pipeline architecture to only 15 seconds. It is thus expected that at least one order of speed gain can be obtained by the parallel architecture and two orders by the pipeline architecture. Though the pipeline architecture is much faster, it is more difficult to implement in view of the multi-reading system, and the more complicated multiplier updating mechanism and sequence control. The parallel design stores multipliers locally, and requires simple hardware to read and update multipliers. Further study is underway to finalize the selection.

		Parallel	Pipeline	LR
One iteration	Clock cycle	$I \times J \times K$	$I \times K$	
	Second	0.5	0.025	6
600 iteration		5 min.	15 sec.	1 hour

Table 2: Speed comparison of LRNN with LR.

5 SUMMARY

A parallel architecture and a pipeline architecture are designed to implement LRNN, with their performance analyzed. Such hardware implementations are expected to reduce the computation time by one to two orders of magnitude, and to provide an on-line, near-optimal scheduling system to be embedded within a PC.

6 ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation grant DMI-9813176 and by the National Science Council of Taiwan, Republic of China, under grant NSC-87-2622-E-002-011.

7 REFERENCES

[1] Zijm, W. H. M, Kals, H. J. J., 1995, Integration of Process Planning and Shop Floor Scheduling in Small Batch Part Manufacturing, Annals of the CIRP, 44/1:429-432.

[2] Foo, Y. P. S., Takefuji, Y., 1988, Integer-linear Programming Neural Networks for Job-shop Scheduling, Proc. of the IEEE 2nd International Conference on Neural Networks, pp. 341-348.

[3] Hopfield, J. J., Tank, D. W., 1985, Neural Computation of Decisions in Optimization Problems, Biological Cybernetics, 52:141-152.

[4] Luh, P. B., Wang, J. H., Wang, J. L., Tomastik, R. N., 1997, Near Optimal Scheduling of Manufacturing Systems with Presence of Batch Machines and Setup Requirements, Annals of the CIRP, 46/1:397-402.

[5] Luh, P. B., Zhao, X., Wang, Y., Thakur, L.S., 1998, Lagrangian Relaxation Neural Network for Job Shop Scheduling, Proc. of International Conference on Robotics and Automation, Leuven, Belgium, pp. 1799-1804.

[6] Wang, J., Luh, P. B., Zhao, X., Wang, J., 1997, An Optimization-Based Algorithm for Job Shop Scheduling, SADHANA, 22/2:241-256.

[7] Weste, N. H. E., Eshraghian, K., 1993, Principles of CMOS VLSI Design: a System Perspective, Addison-Wesley, MA.

[8] Willems, T. M., Brandts, L. E. M. W., 1995, Implementing Heuristics as an Optimization Criterion in Neural Networks for Job-shop Scheduling, Journal of Intelligent Manufacturing, 6/6:377-387.

[9] Wilson, V., Pawley, G. S., 1988, On the Stability of the Traveling Salesman Problem Algorithm of Hopfield and Tank, Biol. Cybernetics, 58:63-70.

[10] Zhou, D. N., Cherkassky, V., Baldwin T. R, Olson, D. E., 1991, A Neural Network Approach to Job-shop Scheduling, IEEE Trans. Neural Networks, 2/1:175-179.