SIMD ARCHITECTURE FOR JOB SHOP SCHEDULING PROBLEM SOLVING

Kuan-Hung Chen[1], *Shi-Chung Chang*[1], *Tzi-Dar Chiueh*[1], *Peter B. Luh*[2], *Xing Zhao*[2]

[1] Department of Electrical Engineering, Room 511 National Taiwan University, Taipei, Taiwan 10617
[2] Department of Electrical and Computer Engineering, University of Connecticut
Storrs, Connecticut 06269-2157, USA

ABSTRACT

Job shop is a typical environment for manufacturing high-variety and low-volume discrete parts. Good scheduling is critical and challenging to the competitiveness of job shops. The Lagrangian relaxation neural network (LRNN) developed by [1], provides an approach of quantifiable quality and successful industrial applications. To further speed up scheduling for large-scale problems, in this paper, the parallelism of the LRNN approach is exploited for hardware implementation. New designs include a SIMD architecture, its associated instruction set and detailed circuits. Logic level simulation of the circuit design shows consistent schedules with those obtained by a software implementation. The hardware implementation is expected to have a one to two orders speed-up over the software one.

1. INTRODUCTION

Job shop is a typical environment for manufacturing low-volume and high-variety discrete parts. In a job shop, parts with various due dates and priorities are to be processed on various types of machines. Job shop scheduling selects machines and beginning times for processing individual operations to achieve certain objectives under the given machine capacity and computation time constraints. A good solution to scheduling problems can result in significant savings. For example, a scheduling system developed by IBM-Japan is estimated to save over a million dollars a year for a major steel company [2].

There are two main challenges for effective scheduling: solution quality and solution finding speed. Theoretically, computation complexity of many job shop scheduling problems is NP [3]. The generation of an optimal schedule often requires excessive computation time regardless of the methodology. Instead, near- or sub-optimal solutions are adopted for practical applications and there have been many sub-optimal or heuristic methods [3].

Recently, there have been a series of scheduling methods with successful industrial applications [4] [5] developed under a common framework of Lagrange relaxation (LR). These methods relax the coupling constraint(s) of a scheduling problem by applying the Lagrangian relaxation technique. The original scheduling problem is then decomposed into independent, simpler optimization subproblems and a Lagrange multiplier optimization problem. Various optimization techniques are developed for efficient solution with quantifiable optimality. To further advance the computation efficiency of this class of methods, Luh et al. exploited the inherent parallelism of their LR-based job shop scheduling methods and designed a LR neural network (LRNN) algorithm [1] for parallel computing.

In this paper, we further enhance the parallelism in LRNN and design parallel processing hardware for the parts of intensive computations in LRNN to speed up the computation. We first analyze and modify the LRNN scheduling algorithm to be amenable for parallel hardware implementation. We then design for the algorithm a parallel processing architecture and its associated instruction set. Finally, logic circuit implementation is designed, simulated, and then fabricated. The performance of our design is demonstrated by the preliminary testing result of the hardware implementation.

The remainder of the paper is organized as follows. In section 2, the problem formulation and the LRNN algorithm are presented. Section 3 describes a SIMD architectural design for the VLSI chip implementation. Section 4 presents two key modules of our circuit design, Verilog simulation results and the estimation of speedup. Finally, a summary of our design is given in Section 5.

2. MODIFIED LRNN ALGORITHM

2.1. Problem Formulation

Consider a job shop, where there are H machine types and each machine type may consist of a few identical machines. There are I parts to be scheduled over a time horizon of K time units. Part i has its due date D_i, weight (or priority) factor W_i, and requires J_i sequential processing operations. Each operation requires the processing by a machine of a specific type for pre-specified units of time. Processing of each operation must satisfy the operation precedence constraint, i.e., its processing may start only after the completion of its preceding operation. The number of operations assigned to machine type h at time k should not violate the machine capacity constraint and it should no more than the number of machines available at that time, M_{kh} , i.e.,

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

where δ_{ijkh} is a 0-1 variable and equals 1 if operation j of part i is being processed by machine type h at time k; it equals 0 otherwise. Variables δ_{ijkh} are determined once the beginning times b_{ij} of all operations are decided.

The scheduling goal of on-time delivery for individual jobs is modeled as penalties on job delivery tardiness $T_i = \max[0, C_i - F_i]$, where C_i is the completion time of part i and is equal to the beginning time of the last operation of part i plus its processing

This work was supported in part by the National Science Council of Taiwan, Republic of China, under grant NSC-89-2212-E-002-040 and by the National Science Foundation of the United States of America under grant DMI-9813176.

0-7803-6685-9/01/$10.00©2001 IEEE

time. The scheduling problem then boils down to determine beginning times b_{ij} of individual operations to minimize the total weighted part tardiness $\sum_i W_i \times T_i$ while satisfying all the constraints. With linear constraints and additive objective function, mathematically, this formulation is a *separable* optimization problem since all the constraints are linear and the objective function is additive.

2.2. Solution by LRNN

Now apply Lagrangian relaxation to machine capacity constraints and obtain a "relaxed problem" as

$$\min_{\{b_{ij}\}} L,\ withL = \sum_i W_i T_i + \sum_{kh} \left(\pi_{kh} \left(\sum_{ij} \delta_{ijkh} - M_{kh} \right) \right) \tag{2}$$

subject to individual part constraints,

where π_{kh} are the Lagrange multipliers. By exploiting the separability, the relaxed problem can be decomposed into the following decoupled part subproblems for a given set of multipliers:

$$\min_{\{b_{ij}\}} L_i,\ withL_i = W_i T_i + \sum_{j-1}^{J_i} \sum_{k=b_{ij}}^{c_{ij}} \pi_{kh_{ij}}, i = 1, \ldots, I, \tag{3}$$

subject to operation precedence constraints of part i,

where h_{ij} denotes the machine type used by operation j of part i. Each δ_{ijkh} in (2) is set, in the decomposition, to a value 0 or 1 according to its definition.

Let L_i^* denote the minimal subproblem cost of part i under the given multipliers. A dual problem is then obtained as

$$\max_{\{\pi_{kh}>0\}} D,\ withD = \sum_i L_i^* + \sum_{kh} (M_{kh} \times \pi_{kh}). \tag{4}$$

The dual function D is concave and provides a lower bound to the original scheduling problem. Interested readers may refer to [1] for details.

A surrogate subgradient method is adopted to solve the dual problem in (4). Under a given set of Lagrange multipliers, π_{kh}, part subproblems can be solved independently among parts. After solving a part subproblem and obtaining the beginning times of individual operations, this method updates the subgradient of the dual function D based on the solution of the subproblem. One iteration of the method consists of solving all part subproblems and updating the corresponding subgradient once. The procedure iterates until a convergent dual solution is obtained. As there may be machine capacity violations in the dual solution, a heuristic then adjusts it to a feasible one. Solving a part subproblem is the most computation intensive of all.

S-NBDP for Solving Subproblems

Each part subproblem (3) is a multistage optimization problem. We design a simplified neuron-based dynamic programming (S-NBDP) algorithm for its solution, which combines the ideas of NBDP [6] and SDP [7] with consideration of hardware implementation. The basic structure of the S-NBDP application to a part subproblem is depicted in the dash-lined box of Fig. 1, where state and comparison neurons perform backward DP [8] computation while a forward sweep procedure identifies the optimal schedule from results of the backward DP. Neurons are connected based on precedence constraints. In Fig. 1, an arrow indicates the direction of data flow.

In the backward DP procedure for a part subproblem, a stage corresponds to an operation and a state corresponds to an operation beginning time. The backward DP is a stage-by-stage iterative procedure starting from the last stage. In stage j, all state neurons of the stage computes in parallel their respective cumulative cost by adding up the stage-wise cost and the optimal cost-to-go (OCTG) of the state. The stage-wise cost of a state is the summation of multipliers associated with the machine type needed during the processing time of the stage (operation). The OCTG of a state represents the minimum cost to schedule the remaining part operations after the state (time). For a state in the last stage, the OCTG equals to the tardiness penalty of the state.

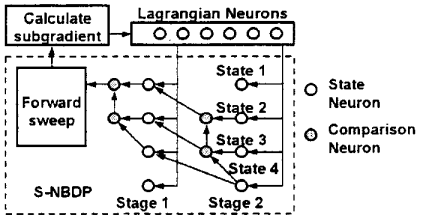

Figure 1: LRNN Structure.

Comparison neurons then find the OCTGs for individual states of the preceding stage, i.e., stage $j-1$. In the procedure, one comparison neuron (CN) inputs the output of another CN. For the example in Fig. 1, the CN for OCTG of state k compares and finds the smaller value between the cumulative cost of state k and the output of a CN for state $k+1$. Such a comparison procedure is obviously sequential starting from the last state. However, software simulation results show that beginning time of an operation obtained in one LRNN iteration usually differs from that of the previous iteration by only few time units. It in turn suggests that only OCTGs of some adjacent states in a stage may actually need to be calculated. Such an observation motivates our simplification of the comparison procedure.

In the simplified comparison procedure for a stage, a set of adjacent states is first identified based on beginning time obtained from the previous iteration. Each CN for a state in the set performs comparison normally as described in the previous paragraph. For a state not in the set, the CN does nothing but relaying the cumulative cost of the state. For each comparison neuron, there is a state flag to record which comparison operand is the minimum. In the case of Fig. 1, a flag value '1' indicates that the cumulative cost of current state is the minimum and '0' otherwise. Computations by both the state and comparison neurons are functionally repetitive from one stage to the next.

New beginning times of individual operations are then identified from state flags of all stages by a *forward sweep* procedure. It searches through state flags stage by stage starting from the first stage. Within a stage, the search is done state-by-state starting from the state corresponding to the earliest beginning time of that stage (operation), which is equal to 1 plus the completion time of previous stage. Whenever a flag of value 1 is found at a state of a stage, the time (state) is set as the beginning time of the operation (stage). The *forward sweep* completes S-NBDP for a part.

Sugradient and multiplier updating

After solving a part subproblem in one iteration, the difference of machine usage at each time between schedules resultant from the current and previous iterations, $diff_{kh}$, is calculated. Each element of the subgradient g of the dual function D is updated by $g_{kh}^{(n+1)} = g_{kh}^{(n)} + diff_{kh}^{(n)}$, where $n = 1, 2, \ldots$, is the iteration index and $g_{kh}^{-1} = -M_{kh}$. Multipliers are then updated according to the formula, $\pi_{kh}^{(n+1)} = \pi_{kh}^{(n)} + \alpha_{kh}^{(n)} \times g_{kh}^{(n)}$, by Lagrangian neurons, where α is the step size parameter. Note that both updating can be done in parallel among different (k, h) pairs.

3. SIMD ARCHITECTURE

To exploit the parallelism of LRNN, a system as shown in Fig. 2 is designed, which consists of a PC, a micro-controller, and several LRNN chips. Software in the PC takes problem inputs, structures the data and generates controlling commands to the micro-controller. The micro-controller then feeds the data from PC to individual LRNN chips, controls their processing sequences, and returns their solutions to PC for output. Each LRNN chip implements, with a limit on problem dimension, S-NBDP and the surrogate subgradient updating of multipliers. Chips can be cascaded for various problem dimensions. Based on the dual solution obtained by LRNN chips, the software in PC performs feasibility adjustment of the final solution.

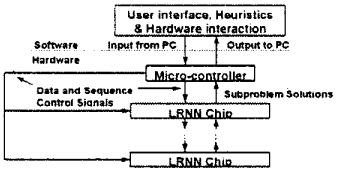

Figure 2: Overall system architecture.

In this system, the LRNN chips carry out most of the computation for finding a dual optimum. A SIMD architecture shown in Fig. 3 is designed to map the modified LRNN into a parallel hardware implementation. Under this architecture, an instruction decoder decodes the instruction code from the micro controller into control signals for the whole chip. Processing elements (PEs) carry out many arithmetic operations in parallel such as the addition in calculating the cumulative cost by each State Neuron, the comparison by each Comparison Neuron, and the updating of Lagrange multipliers. The parallelism of the first two types of operations is quite straight forward based on the modified LRNN structure and it is natural to have one PE to support each state and CN pair. As for multiplier (subgradient) updating, its parallelism is rooted in that one multiplier (gradient) is defined for each machine type at each time period. Since a state corresponds a time unit, it becomes obvious that the PE for the state can be used to store all the multipliers (gradients) of the corresponding time period, i.e., g_{kh}, π_{kh}, for all h. Similarly, the calculation of subgradient direction and the updating of Lagrange multipliers by the Lagrangian Neuron can also be carried out by individual PEs. A forward sweep circuit (FSC) then reads state flag values from PEs and performs a sequential search to find out the beginning times of individual operations. The global memory stores input data items such as due dates and output data items such as operation beginning times.

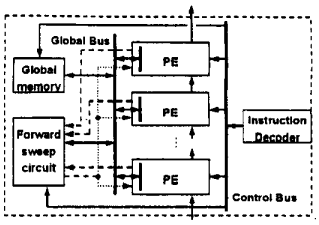

Figure 3: SIMD architecture of a LRNN Chip.

4. CIRCUIT DESIGN

Designs of PE and FSC are more involved than those of the global memory and the instruction decoder. Key design concepts of the two are described as follows.

4.1. PE Design

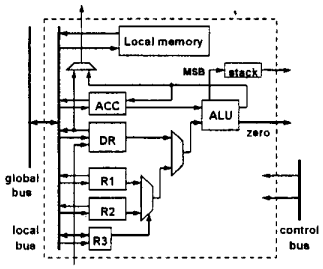

Figure 4: PE architecture.

Fig. 4 depicts our circuit architecture design of PE. To perform arithmetic operations required by the LRNN algorithm, a standard circuit design for arithmetic logic unit (ALU) is adopted. The ALU is capable of performing addition, comparison, etc. In executing most instructions, inputs to ALU are latched by two registers, the ACC (accumulator) and the DR (data register). Registers R1, R2, and R3 latch data for arithmetic operations conditioned on a preceding ALU execution result. The local memory is required to store the Lagrange multipliers and the machine usage information for each machine type. A local bus communicates between local memory and registers such as ACC, DR, etc. within a PE. It also serves the global data communications with the global bus. Finally, instead of using a 16-bit storage unit in the local memory to store each 1-bit state flag, we design a stack for string state flags.

4.2. Forward Sweep Circuit

The **forward sweep circuit** implements *forward sweep* steps for state flag search within a stage. Each state has a logic unit with three 1-bit inputs: a **begin flag** with '1' indicating that this state is the earliest beginning state for the search within this stage or '0' otherwise, the **state flag** of this state, and an **input search flag**. The logic unit has two 1-bit outputs: an **output search flag** feeding to the next state as its input search flag and a **time flag** indicating the new beginning time. The search is a sequential procedure and always starts at the first state and completes at the last states. The search flag, that is initially '0', will be set to '1' at the earliest

beginning state. Time flag of the logic unit is set to '1' *iff* state flag of value '1' is found at this state when the input search flag also has a value '1', and the search flag is reset to '0'. Such a sequential search will require a long search time for a large-scale problem. Note that the output search flags should be 0 for states before the earliest beginning state and for states after the state where its time flag is set to '1'. So we adopt an idea similar to the "carry bypass" concept of the Manchester carry chain [9] to speed up the search.

4.3. Experimental Results

Our circuit design is finished with 16 PEs in one chip. A test problem of 5 parts, 2 machine types and a time horizon of 16 time units is used for logic level simulation. Verilog is adopted as the logic simulator. A software implementation of our modified LRNN algorithm in C code serves as the benchmark. In specific, values of Lagrange multipliers obtained by the C code and by the Verilog simulation are compared. Tables 1 and Fig. 5 give the results respectively. Comparing the two sets of results, we clearly see that they are all the same. This is a justification of logical correctness of our circuit design.

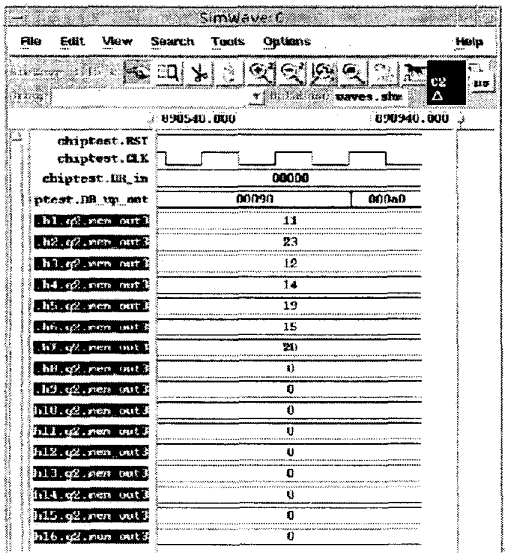

Figure 5: Multipliers of type-1 machine generated from Verilog simulator.

The proposed architecture has been fabricated by Taiwan Semiconductor Manufacturing Co., using a single-poly quadruple-metal 0.35-μm CMOS technology. The chip measures 4.56 mm $\times$ 4.24 mm. There are 16 processing elements and 356k transistors in the chips. Preliminary testing results show that the chip works at a clock rate of 100 MHz while drawing only 742 mW from a 3.3V power supply. Under the above operating condition, 0.15ms is required to complete 10 iterations for the previous 5-part test problem. The software solution takes approximately 3.5ms on a K6-II 300MHz computer. More than 20 times of speed up is achieved. The gain is derived from the parallel computation by 16 PEs and the simplification of the sequential pair-wise comparisons. With putting more PEs in one LRNN chip and cascading the LRNN chips, two orders speed-up may be achieved for larger problems.

Table 1: Multipliers of type-1 machine from C program.

	Multipliers		Multipliers
State 1	11	State 9	0
State 2	23	State 10	0
State 3	12	State 11	0
State 4	14	State 12	0
State 5	19	State 13	0
State 6	15	State 14	0
State 7	20	State 15	0
State 8	0	State 16	0

5. SUMMARY

In this paper, a SIMD architecture has been designed to implement the modified LRNN algorithm for speeding up the job shop scheduling problems solving. The design concepts of the architecture and the circuit are presented. The logic-level simulation results show the feasibility of the SIMD architecture to implement the modified LRNN algorithm with parallel hardware efficiently. The preliminary performance testing of hardware implementation demonstrates the potential of one to two orders speed-up over the software one.

6. REFERENCES

[1] P. B. Luh, X. Zhao, and Y. Wang, "Lagrangian Relaxation Neural Networks for Job-Shop Scheduling," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, pp. 78–88, February 2000.

[2] M. Numao and S. Morishita, J. Kittler, "A Scheduling Environment for Steel-making Processes," in *Proceedings, Fifth Conference, Artificial Intelligence for Applications*, pp. 279-286, 1989.

[3] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Inc., 1995.

[4] Liao D.Y., Chang S.C., Pei K.W., and Chang C.M. "Daily scheduling for R&D semiconductor fabrication". IEEE Transactions on Semiconductor Manufacturing. Vol. 9, no. 4, Nov. 1996, pp. 550 -561.

[5] Hoitomt D.J., Luh P.B., and Pattipati K.R. "A practical approach to job shop scheduling problems". IEEE Transactions on Robotics and Automation. Vol. 9, Feb. 1993, pp. 1-13.

[6] Luh P.B., Zhao X., Thakur L.S., Chen K.H., Chiueh T.D. and Chang S.C. "Architectural Design of Neural Network Hardware for Job Shop Scheduling". Annals of the CIRP. Vol. 48/1, 1999, pp. 373-376.

[7] X. Zhao, K. H. Chen, P. B. Luh, T. D. Chiueh, S. C. Chang and L. S. Thakur, "Integrated online job shop scheduling system," *SPIE International Symposium on Intelligent Systems and Advanced Manufacturing. Boston, MA*, 1999.

[8] Dimitri P. Bertrekas, *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., 1987.

[9] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*. Addison Wesley, 1992.