by David Landén
7812292436
AIICS, IDA, Linköpings Universitet.
davla@ida.liu.se

Summary of ”A Spatial Path Scheduling Algorithm for EDGE Architectures”

This report is a summary of the paper ”A Spatial Path Scheduling Algorithm for EDGE Architectures” [1] presented at the paper presentation session in the Advanced Parallel Programming course.

Introduction - the Problem

According to the paper, the on-chip wire delays are and will be of greater importance in future architectures when moving onward to more and more complicated processor architectures. One solution to this problem is to let the compiler handle the communication issue among functional units as well. One example of such an ”communication-exposed microarchitecture”[1] suitable for this approach is the EDGE architecture. The EDGE consists of a grid of ALUs, connected to register banks and data cache banks. An instruction executed on one ALU pushes the result to any nearby ALU and this reduces the number or stores to registers that are needed. The focus of the EDGE architecture is on concurrency within a single thread, i.e. concurrency on the instruction level. The problem then becomes how to map a block of instructions to the grid in such a way that execution time is minimized and instruction level concurrency is increased.

A new compiler scheduling algorithm, called Spatial Path Scheduling, that provides a solution to this problem, is described and evaluated. The algorithm does not provide an optimal solution for the scheduling problem, that is not practical possible, due to all possibilities for the scheduling. Instead experiments using simulated annealing to minimize the critical path were carried on programs from various test benches. The results from the experiments was then used to enhance their own scheduling algorithm with heuristics that mimic the outcome from the experiments.

EDGE Architectures

The EDGE architecture used in the experiments and described in the paper [1] is called TRIPS [2]. It has a 4*4 array of ALUs (see Figure trips). Each ALU has eight issue slots, meaning that a block of maximum 128 instructions can be mapped to the grid at a time. The loading of instruction to the grid has a fixed cost, therefore it is important to try to use as full blocks as possible. The TRIPS architecture is a SPDI architecture, meaning that instructions are statically placed (by the scheduler), dynamically issued (i.e. the instructions are executed when their operands arrive). The TRIPS model can be compared to VLIW, that are statically placed, statically issued (SPSI) and out-of-order super scalar architectures that are dynamically placed, dynamically issued.

The Spatial Path Scheduling Algorithm

The Spatial Path Scheduling (SPS) algorithm maps dataflow graphs to the ALU topology. The algorithm concentrates on the critical path, that is the longest chain of instructions in the grid. By minimizing this path, i.e. placing the instructions in a clever way, the total execution time of the block will be minimized. The instructions can not be placed in any order on the grid. A chain typically starts in a register, meaning that there are some fixed point to take into account for the scheduler, called anchor points in the SPS algorithm. When an instruction is placed, then it also become an anchor point. The instructions that are possible candidates to be scheduled next are listed in the open list, only instructions which parents already has been scheduled or those that do not have any input belong to this list.

SPS works as follows:

1. For each instruction in the open list, compute all legal locations for it.

2. For each location slot, computes a placement cost that is the cost to place a given instruction in that slot.

3. SPS schedules the instruction, whose lowest placement cost is the highest of the lowest placement cost across all instructions in the open list. Any missing children of the scheduled instruction are added to the open list.

4. Repeat from 1, until finished.

There are three problem with the base SPS algorithm. It does not care about contention on the local or global level or on the network. Since it only schedules one block at a time, it has a local view of the most critical path. The third problem is that it ignores delays that occurs when the number of instruction are greater than the instruction capacity of the ALUs along a path.

By studying the result from the simulated annealing experiments, the SPS algorithm's placement cost function (from step 2 in the algorithm description) could be changed to take the three problems into account. An utilization penalty is added to the placement cost function. This penalty consists of various parts and the first part is a penalty cost that is added when placing a new instruction on an ALU where instructions already are scheduled (possible resource conflict). A link contention penalty is added for placing an instruction, where the penalty is related to the number of network links that are consumed by that instruction. The global ALU contention is given by adding the sum of all issue slots consumed by all instructions scheduled on the same ALU to the placement cost for an instruction on that slot.

The cost parameters are weighted by the fullness relation and critical path ratio. The fullness relation says how much of the grid that is filled (i.e. how many of the 128 instructions slots that are taken), the criticality is the ratio between the path containing the instruction we are about to schedule path and the critical path. Since it was shown from the simulated annealing results, that blocks with very high concurrency seldom had any clear critical path, this fact was added to the criticality relation.

criticality = (pathLength/criticalPathLength)/concurrency.

It was also noted that full blocks depends more on good link utilization than on ALU utilization, and this was weighted by the fullness relation in the utilization penalty.

The penalty then becomes:

utilPenalty = localALUCntn + globalALUCntn * (1- fullness)*(1-criticality) + globalLinkCntn*fullness*(1-criticality)

And the new placement cost function is:

pCost(i,slot) = inputLatency + utilPenelty + execLatency + outputLatency + additionalRoutingCost.

The additionalRoutingCost depends on the amount of instructions left to schedule and the possible issue slots to next anchor point. The input latency is the time when the last input operand arrives and the instruction i is ready to fire. The execLatency is the number of cycles necessary to execute i with no contention. The outputLatency is the maximum expected number of cycles from i to any output-producing leaf instruction.

Results and Conclusion

It was shown that the full SPS algorithm achieved a 21% improvement (on the average) over the previous best scheduler for this architecture. The result was within 5% of the annealed results. It was also shown that the three heuristics worked better together than one by one. Alone they provided less than 4% improvement but all together increased the improvement to 7%.

References

[**1**] Katherine E. Coons, Xia Chen, Sundeep K. Kushwaha, Doug Burger, Kathryn S. McKinley "A Spatial Path Scheduling Algorithm for EDGE Architectures",
http://www.cs.utexas.edu/users/cart/trips/publications/asplos06.pdf

[**2**] Doug Burger, Stephen W. Keckler, Kathryn S. McKinley, Mike Dahlin, Lizy K. John, Calvin Lin. Charles R. Moore, James Burrill, Robert G. McDonald, William Yoder, and the TRIPS Team "Scaling to the end of silicon with EDGE architectures",
http://www.cs.utexas.edu/users/asmith/pubs/IEEECOMPUTER04_trips.pdf