Mini-task report: SDC with simulated annealing

Ludwig Meysel, Mitja Stachowiak

TECHNISCHE
UNIVERSITÄT
DARMSTADT

1 Introduction

The task was to implement a simulated annealing approach (SA) for SDC (system of difference constraints). LPsolve is used to get a schedule for a given set of constraints. The SA-algorithm mutates the order of the constraints to reduce the number of clock cycles of the schedule.

2 Resources and Constraints

The constraints consist of data flow - and resource constraints. The data flow constraints determine, that no operation must start, before all predecessors have obtained a result. The resource constraints prevent, that the same resource is used twice at the same time.

2.1 Resources

Each hardware has a certain amount of resources and resource types. There is a fixed list of operations given in the framework:

Operation name (ar)	delay	weight
MEM	2	9.0
ADD	1	1.0
SUB	1	1.4
MUL	4	2.3
DIV	18	4.3
SH	1	2.0
AND	1	2.0
OR	1	2.0
CMP	1	2.1
OTHER	1	1.0
SLACK	1	0.0

Each resource type can support multiple operations. For this project, the resource(types) are assumed to be overlap-free:

$$\neg\exists R_1, R_2 \in Resources; Op_1, Op_2 \in Operations : Op_1 \in R_1 \wedge Op_1 \in R_2 \wedge Op_2 \in R_1 \wedge Op_2 \notin R_2$$

Each resource can handle one operation within a certain time (delay). Multiple resources of the same type may exist.

2.2 Data Flow Constraints

The data flow constraints are fixed and only need to be computed once.

2.3 Resource Constraints

The data flow constraints are fixed and only need to be computed once.

3 Simulated Annealing

The principal structure of any simulated annealing looks like this:

```
S = RandomConfiguration();
T = InitialTemperature();
while (ExitCriterion()==false) {
  while (InnerLoopCriterion() == false) {
    S_new = Generate(S);
    Δ C = Cost(S_new)-Cost(S);
    r = random(0,1);
    if (r < e^(-ΔC/T)) S = S_new
  }
  T = updateTemperature();
}
```

The implementation is located in scheduler/SASDC.java:schedule. The parameters are:

- *Random Configuration* ...
- *Initial Temperature* is determined by applying n(nodes) random changes and saving the costs of each change. T is then $20 * standardDeviation(costs)$.
- *Exit Criterion* is the condition, when the simulated annealing should stop. For each temperature, the number of applied changes and the number of accepted changes is counted. When less then 12% of the changes are accepted, the algorithm stops.
- *Update Temperature* decreases T by a factor tu, which depends on the acceptance ratio as well:

acceptance ratio (ar)	temperature factor (tu)
> 96%	0.5
96 .. 80%	0.9
80 .. 15%	0.95
< 15%	0.8

- *Inner Loop Criterion* determines, how many changes are tested for the same temperature. Each change usually moves one node in the ordering of constraint-equations. The larger the number of nodes becomes, the more often each node should be moved, so the number of iterations should depend on the node count. Further more, there is a quality factor $\in [1..10]$ for the algorithm, which can be passed via the third program argument. The formula $n_{inner} = \left\lceil quality * n_{nodes}^{4/3} \right\rceil$ is known to yield a result, thats quality belongs to the given quality.

ADPCMn-decode-271-381.dot	27	64	80	56	0	31	90	1	20494	29	39
ADPCMn-decode-271-381.dot	27	64	80	56	0	30	50	5	82216	113	38
ADPCMn-decode-271-381.dot	27	64	80	56	0	30	50	10	144181	185	55
ADPCMn-decode-425-472.dot	12	23	80	23	80	34	20	1	85	0	11
ADPCMn-decode-425-472.dot	12	23	80	23	80	21	80	5	7591	6	37
ADPCMn-decode-425-472.dot	12	23	80	23	80	21	80	10	11001	9	13
ADPCMn-decode-524-553.dot	7	25	40	26	40	25	40	1	15	0	2
ADPCMn-decode-524-553.dot	7	25	40	26	40	25	40	5	68	0	7
ADPCMn-decode-524-553.dot	7	25	40	26	40	25	40	10	135	0	13
ADPCMn-decode-559-599.dot	9	26	10	27	20	28	20	1	20	0	3
ADPCMn-decode-559-599.dot	9	26	10	27	20	25	10	5	4137	3	26
ADPCMn-decode-559-599.dot	9	26	10	27	20	28	20	10	377	0	35
ADPCMn-decode-631-729.dot	23	44	80	40	0	33	30	1	5017	5	81
ADPCMn-decode-631-729.dot	23	44	80	40	0	24	30	5	13121	14	65
ADPCMn-decode-631-729.dot	23	44	80	40	0	22	90	10	108076	120	56
ADPCMn-decode-771-791.dot	5	13	50	13	50	22	50	1	10	0	1
ADPCMn-decode-771-791.dot	5	13	50	13	50	22	50	5	44	0	4
ADPCMn-decode-771-791.dot	5	13	50	13	50	22	50	10	87	0	8
ADPCMn-decode-803-832.dot	8	18	80	18	80	20	20	1	17	0	2
ADPCMn-decode-803-832.dot	8	18	80	18	80	20	20	5	81	0	8
ADPCMn-decode-803-832.dot	8	18	80	18	80	20	20	10	161	0	16
AESrkgcyclic.dot	24	24	40	23	40	37	40	1	421	0	61
AESrkgcyclic.dot	24	24	40	23	40	32	40	5	13881	18	84
AESrkgcyclic.dot	24	24	40	23	40	30	40	10	28414	38	14
BLAKE256Digest-processBlock-160-230.dot	21	57	40	58	40	32	40	1	10905	12	11
BLAKE256Digest-processBlock-160-230.dot	21	57	40	58	40	22	40	5	58291	64	11
BLAKE256Digest-processBlock-160-230.dot	21	57	40	58	40	31	40	10	175161	194	6
BLAKE256Digest-processBlock-189-1577.dot	308	414	90	128	70	110	90	1	6244	241	63
BLAKE256Digest-processBlock-189-1577.dot	308	414	90	128	70	97	90	5	31204	1236	87
ContrastFilter-filter-13-252.dot	47	72	0	67	70	50	80	1	4251	9	49
ContrastFilter-filter-13-252.dot	47	72	0	67	70	37	70	5	24622	54	0
ContrastFilter-filter-13-252.dot	47	72	0	67	70	33	20	10	101821	218	61
ECOH256Digest-AES2RoundsAll-2-666.dot	179	262	30	174	40	58	10	1	175567	2733	60

5 Conclusion