NTNU – Trondheim
Norwegian University of
Science and Technology

A Parallel Solution to Large Scale Hydropower Scheduling

By reducing the number of synchronization points in the Stochastic Dual Dynamic Programming (SDDP) Algorithm

Hallvard Braaten

Master of Science in Physics and Mathematics
Submission date: October 2014
Supervisor: Trond Kvamsdal, MATH
Co-supervisor: Arne Morten Kvarving, IMF
Arild Helseth, SINTEF

Norwegian University of Science and Technology
Department of Mathematical Sciences

ABSTRACT

Stochastic dual dynamic programming (SDDP) has become a popular algorithm used in practical long-term scheduling of hydro power systems. The SDDP algorithm is significantly more computationally demanding than most heuristic-based scheduling methods, but can be designed to take advantage of parallel processing. This thesis presents a novel parallel scheme for the SDDP algorithm, where the stage-wise synchronization point traditionally used in the backward iteration of the SDDP algorithm is either partially or fully relaxed. The proposed scheme was tested on a realistic model of a Norwegian water course, proving that the partial synchronization point relaxation significantly improves parallel efficiency.

SAMMENDRAG

Stochastic dual dynamic programming (SDDP) har blitt en populær algoritme til bruk i langtidsplanlegging innen vannkraft. SDDP-algoritmen er betydelig mer ressurskrevende enn de fleste heuristikk-baserte planleggingsmetoder, men algoritmen er egnet for parallellprosessering. Tradisjonell parallellisering av denne algoritmen har synkronisering mellom hvert tidssteg i bakoveriterasjonen. Den foreslåtte måten å parallellisere på fjerner denne synkroniseringen helt eller delvis. Den er testet på en realistisk modell av et norsk vassdrag, og resultatene viser at ved å delvis fjerne synkroniseringspunktene øker den parallelle effektiviteten betydelig.

ACKNOWLEDGEMENTS

Many thanks to my supervisor Arild Helseth who has guided and helped me through this entire project, and my supervisors at NTNU Trond Kvamsdal and Arne-Morten Kvarving who have helped with computer-related and administrative challenges. I'd also like to thank my highschool teacher Tor Sandnes who triggered my interest in science. I have to thank my fellow students for a great five years at NTNU, and last but not least i have to thank my dear mother, Anne-Marie Riis, who have been my greatest and most important support my entire life.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

LTHS	Long-Term Hydropower Scheduling
SDDP	Stochastic Dual Dynamic Programming
SDP	Stochastic Dynamic Programming
LP	Linear Programming
FCF	Future Cost Function

Part I

INTRODUCTION

The first part aims at giving the reader an understanding of the motivation and the application of hydro power scheduling.

1 INTRODUCTION

The goal of optimal long-term hydro power scheduling (LTHS) is essentially to find the optimal operation strategy, which in this paper means to find the strategy that minimizes the expected cost of external energy sources used to meet a certain demand, while all relevant physical and legislative constraints are met. By operation strategy we typically mean how much water to store in the reservoirs contra how much water to be used for production at any given time. Other factors include unknown future water inflow into the reservoir and variable energy prices. The scheduling period has to be long enough to reflect storage capability of the reservoirs, and the time resolution fine enough to capture the basic hydro system constraints. That is, if the scheduling period is too short the reservoir storage levels are never challenged, and if the time resolution is not fine enough factors such as maximum production levels, variable inflow and energy prices might get too inaccurate. The LTHS problem can be formulated as an optimization problem with three characteristic properties. Firstly, it is dynamic in time due to the ability to store water in hydro reservoirs. That is, there is a link between reservoir discharge decisions taken at a given time and the future cost of operating the system. If more water is used for production today we naturally have less water for tomorrow which means future cost might be higher. Secondly, it is stochastic since the future inflow to reservoirs are unknown. And finally, the hydro system normally involves multiple reservoirs. Since future hydro inflow is unknown, todays optimal decision depends on what might happen in the future. The future also depends on choices made today, as future reservoir levels are influenced by the production amount today.

Numerous solution strategies has been applied to the LTHS problem. One algorithm proposed over 50 years ago, using Stochastic Dynamic Programming (SDP) [16], which fully discretizes the planning horizon in time, different inflow scenarios and various reservoir storages. This causes a so called "curse of dimensionality" which makes computational effort increase exponential with the number of reservoirs. In spite of its shortcoming, models based on SDP are widely used by the power market participants, e.g. in the Nordic power market. These models are based on some kind of reservoir aggregations and depend on heuristics to address the multi-reservoir aspect in a realistic manner.

In order to avoid the dimensionality problem of the SDP algorithm an approach known as stochastic dual dynamic programming (SDDP)

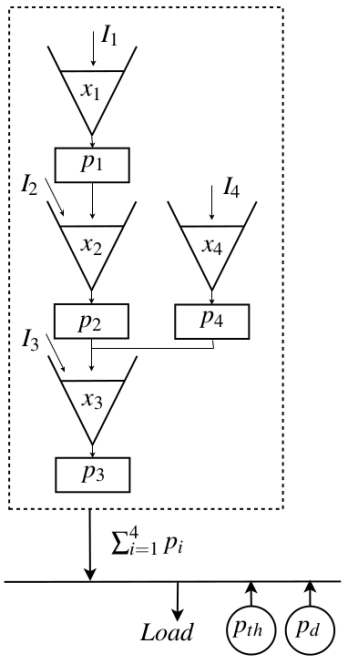


Figure 1: Schematic view of a hydro system.

was presented in [12]. Currently, SDDP seems to be the state-of-the-art method for solving the LHTS problem in regions where hydro power is the dominant technology for production of electric power. In SDDP one avoids having to fully discretize the state space. SDDP is a sampling-based variant of multi-stage Benders decomposition, where an outer future cost function is constructed iteratively for each time-stage by adding Benders cuts. Thus the problem can be decomposed into small linear programming (LP) problems that can be solved independently, which in turn makes it suitable for parallel processing. SDDP is the algorithm used in this paper. An efficient hydro power scheduling algorithm is important because these are used for weekly (sometimes even close to daily) scheduling in hydro power companies today. Thus, it is problematic if the algorithm takes several days to run.

Figure 1 shows a simplified hydrothermal system, consisting of 4 hydro modules, each with a reservoir storage, inflow and production amount. The operation strategy typically consists of deciding how much water that should be used to produce energy at any given time. This means balancing the water levels maintained in the water reservoirs such that spillage and deficit is avoided, while producing energy when the cost of using alternative energy sources is high, as shown in figure 2.

Some parallel implementation of SDDP has already been done, i.e. [15], but in this work there has been looked at a subtle difference. Where [15] synchronized the backward cycle at each stage, the paral-

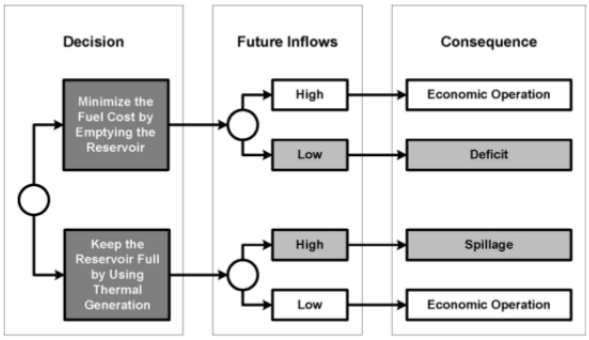


Figure 2: Decision process of hydrothermal systems. [15]

lel algorithm looked at in this paper partially and fully relaxes these synchronization points.

Part II

THEORY

This part gives an introduction to the mathematical theory used in this paper.

2 THEORY

2.1 THE SDDP ALGORITHM

2.1.1 *History*

Initially, hydrothermal operation system planning was solved by SDP (Stochastic Dynamic Programming) [16], where the states variables was fully discretized in time, inflow scenarios and reservoir levels, which causes computational effort to increase exponentially. This has later been addressed by SDDP (Stochastic Dual Dynamic Programming), which introduces sampling to deal with the dimensionality of the stochastic variables.

2.1.2 *Model Description*

The problem consists of a hydro system, which could consists of several modules, each having a reservoir, a power system, and maximum production capacity and a demand of energy within some time frame that has to be met. This energy could be attained either by producing energy in the hydro system by emptying the reservoir or by buying external energy from other energy sources. A planning horizon could span over several years. The wanted result is the production policy that will meet the demand at the lowest expected cost over the planning horizon. Water inflow is unknown and is simulated through some sort of distribution.

Mathematically this can be formulated as

$$\min \mathrm{E}\left\{\sum_{t=1}^{T} \mathbf{c}_t^T \mathbf{x}_t - \boldsymbol{\phi}(V_T)\right\}, \tag{1}$$

where $\mathbf{x}_t$ is a vector of all decision variables at time t, $\mathbf{c}_t$ are a vector of the costs associated with the decision variables. $\boldsymbol{\phi}(V_T)$ is the value of the remaining water in the reservoirs at the end of the planning horizon. The expectation is to be taken over all stochastic variables, which in this paper is only in inflow, but could also be uncertainty in demand and energy prices.

The idea is to discretize the planning horizon into stages, say weeks, and then solve one stage at a time, using the reservoir storage at the end of one stage as the initial reservoir at the next stage. At all future stages water inflow is unknown, so one is interested in testing several different inflow-scenarios as the consequence of a decision at time t depends on the inflow at a later time. For example, if future inflow

is high, one may want to release water to produce energy now, as to avoid spillage at a later stage and thus have wasted potential energy. If future inflow is low, emptying the reservoir now might remove the opportunity to produce energy at a later time when the price of energy is higher. The uncertainty of future inflow must be addressed. This is done by simulating several different inflow-scenarios at each stage (samples generated from a distribution). Thus, the entire problem is divided in both time and inflow-scenarios, as in figure 5, where the sub problem at a given time and inflow scenario is called state.

The sub problem at each state is to minimize the expected total cost, which again can be divided into current cost and future cost, where current cost is the cost of buying external energy to meet the current demand in the period between the beginning of this stage until the end of this stage, while the future cost is the expected cost of buying external energy to meet the current demand from the end of this stage until the end of the planning horizon, given the reservoir levels at the end of this stage. The more water used today means lower cost today, but it also means lower reservoir levels at the end of this stage which again causes higher future cost. The problem is this to find the optimal production level with minimize the total cost.

The future cost as a function of states variable (reservoir levels) at a stage is initally unknown and is piecewise approximated iteratively by simulation for a better and better approximation.

2.1.3 *Dual Dynamic Programming Scheme*

The future cost can as mentioned be piecewise linearly approximated. The idea is the following: We initially run the simulation where the future cost is set to zero. By running the simulation we can find some measure of the value of water at a given stage. Based on these values we give an estimate to the future cost in the *preceding* stage. Then we run the simulation again with a better approximation of the future cost, gain new information of water value, and get even better approximation of the future cost. This continues until convergence is reached.

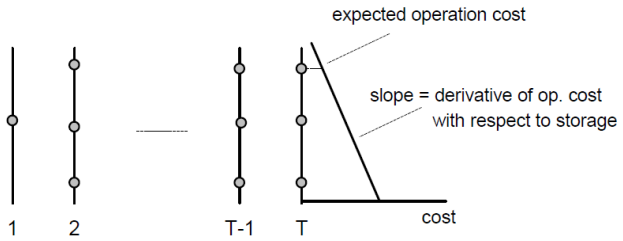


Figure 3: A single cut in the future cost function [12].

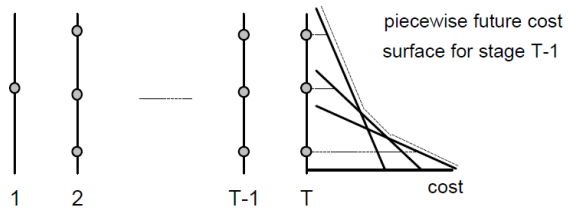


Figure 4: Several cuts of the future cost function [12].

In a simplified scenario with only one hydro module, the future cost function will look something like in figures 3 and 4. The lower the reservoir level is at the end of current week, the more money we have to expect to use to meet future energy demands. Energy prices will increase the more energy is bought, which means that the FCF will not be linear. We can however make a linear approximation, idea illustrated in figures 3 and 4. If we imagine an arbitrary solution in some state with a (thus far) optimal reservoir level x, which will result in the simulation at the cost y. We can get the slope around this point by taking the derivative at this point. By having the slope and a point (x, y), we have one addition to the FCF. This is referred to as a cut. By running several simulations with different inflow scenarios we will get different cuts, and the more cuts we get the better approximation to the FCF we get, as seen in figure 4.

If we have more than one hydro module, all these ideas will be the same but in multiple dimensions.

Because the marginal energy prices is non-decreasing with the amount of energy bought, the FCF is convex. This means that the linear approximation will always be lower or equal to the actual function. This fact is used to get a lower bound on the solution which will be used as a convergence criteria.

The whole problem is solved iteratively, consisting of a forward and a backward cycle, where each iteration gives a better approximation to the FCF. The forward-cycle produces the initial inflow-scenarios and initial reservoirs at all stages, as illustrated in figure 6.

2.1.4 *Forward Cycle*

Each state is solved to find the current optimal solution given the inflow scenario and the cuts created so far, to get the initial reservoir for the next stage. This cycle is also used to find an upper limit Z_{max} for the optimal solution. Assuming equal probability for the different inflow scenarios, Z_{max} is found by taking the weighted sum over the current cost of all stages,

$$Z_{max} = \frac{1}{n} \sum_{i} \sum_{j} \text{current cost in stage } i \text{ and inflow scenario } j,$$

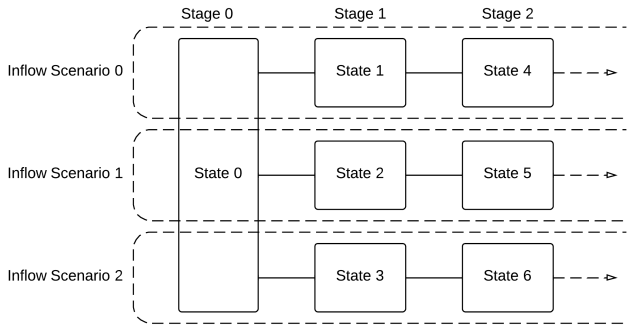


Figure 5: Notation.

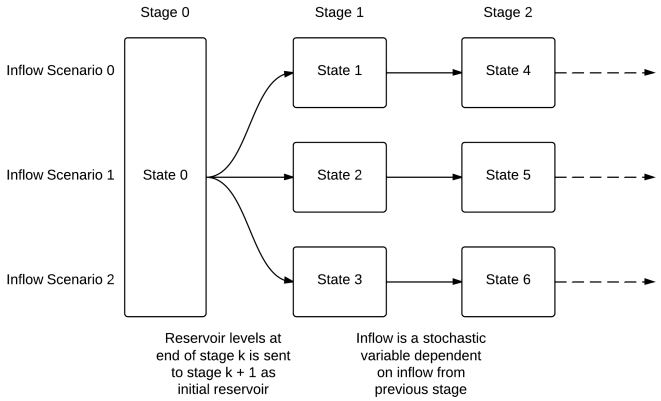


Figure 6: Forward cycle dependency.

where n is the number of inflow scenarios. This is an upper limit as it is a possible solution. However, because of the sampling, there is some uncertainty to this estimation of the upper limit of the total cost. Therefore a confidence interval on this estimation is calculated.

2.1.5 *Backward Cycle*

Backward cycle is used to create the approximation for the future cost function. At each state a Benders cut is created, and used as a constraint in all states at the preceding stage. At each state a fixed number of inflow scenarios are created, where the weighted results of the LP-problems is used to generated one Benders cut. Z_{min} is also found in the backward cycle as the objective value of the optimal solution in the first stage. This is a minimum as the FCF-approximation will always be less than or equal to the true future cost. As in [12], a solution is said to be found when Z_{min} lies inside a 95% confidence interval for Z_{max}.

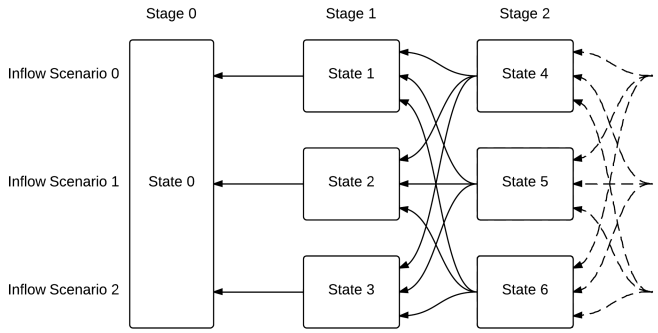


Figure 7: Backward cycle dependency. A Benders Cut is created at each state in stage k and used in all states at stage $k-1$. This is usually referred to as cut sharing and is a central part of the SDDP-algorithm. [9]

2.1.6 *The Scheduling Problem*

2.1.6.1 *Problem formulation*

The scheduling problem can be formulated as described in [8] and restated below:

$$\alpha_t(x_t) = \mathbb{E}_{v_t}\{\min[Z_t(u_t) + \alpha_{t+1}(x_{t+1})]\}$$

Subject to:

$$\begin{aligned} x_{t+1} &= f_t(x_t, v_t, u_t) \\ g_{t+1}(x_{t+1}) &\leqslant 0 \\ h_t(u_t) &\leqslant 0 \end{aligned}$$

where:

$\alpha_t(x_t)$	expected value of operation cost from end of stage t to T assuming optimal operation
$\mathbb{E}$	expected value with respect to all possible inflow sequences v_t
$Z_t(u_t)$	operation cost at stage t associated with decision u_t
u_t	decision variables for stage t
x_t	reservoir storage at the beginning og stage t
v_t	inflow during stage t
T	number of time stages in scheduling period
$f_t(x_t, v_t, u_t)$	state transition equation
$g_{t+1}(x_{t+1})$	constraints on the state variables in stage $t+1$
$h_t(u_t)$	constraints on the decision variables.

$\alpha_t(x_t)$ is the Future Cost Function and is unknown, and is approximated with Benders Cuts. These cuts takes form as additional constraints in the LP-problem. The operation cost $Z_t(u_t)$ is the cost of energy from external energy sources. The decision variables u_t are mainly three for each hydro module. One for the amount of water in

a specific reservoir, one for the amount of water used for production in a reservoir, and one for possible spillage in a reservoir. Some more decision variables are added, which will be addressed later.

Each stage can be divided into even smaller stages, (i.e. weeks divided into days), but where the inflow samples are still generated weekly, such that one LP-problem still is a deterministic problem within a week, and where each day has its own variables. That is, one reservoir has seven variables for each of reservoir storage, production and spillage. One for each day. The modelling presented here is similar to what is formulated in hydro power scheduling tools used by producers in the Nordic power market [5]. See appendix for complete examples of LP-problems.

The slope of a cut is obtained by using the shadow prices of the LP-solution in the state *after* the current state. This means we first have to solve for week two to get a cut for week one. However we first need to solve for week one to get an initial reservoir for week two. This creates a problem where we first solve with no cuts, then create cuts, then solve again with the new cuts, then creates additional cuts, and keep repeating to get a better and better approximation to the FCF and closer to an optimal solution. We say convergence is reached when an upper bound and a lower bound of the optimal solution are sufficiently close. The upper bound of the solution is found by taking the weighted average of the current cost over all states. This is an upper bound because it is a possible solution. And a possible solution have to be an upper bound of the best possible solution. The lower bound is found by taking the objective value at the first stage. This is a lower bound since it is the sum of current cost now and proposed future cost from the FCF which will always be a lower bound to the actual future cost.

2.2 LP-FORMULATION

As described earlier, for a given realization of inflows and a week t, the decomposition in the SDDP algorithm leads to a LP-problem for week t which is formulated as the follows.

2.2.1 *Objective Function*

The objective function, Minimize

$$J = \alpha + \sum_j c_j y_j \tag{2}$$

where α denotes the future expected cost, y_j the amount of external energy bought from thermal unit j, and c_j the cost associated with it. Each thermal unit j has an associated cost and capacity. Thermal

units are used as the source of energy that can be bought to meet the demand.

2.2.2 *Energy Balance*

Energy balance, which is that the sum of total energy produced plus the sum of external energy bought equals the demand.

$$\sum_i p_i x_d^i + \sum_j y_j = d \tag{3}$$

Where x_d^i is the volume of water discharged for module i, p_i is a constant which determines the amount of energy produced per volume water discharged for module i and d is the demand of energy that has to be met.

2.2.3 *Water Balance*

For each hydro module we get a water balance equation which asserts that total inflow of water + initial reservoir levels - discharge of water = remaining reservoir level. The inflow to a hydro module consists of both natural inflow (the stochastic variable) and inflow from the discharge and spillage from hydro modules that are located upstream of this module, as illustrated in the hydro topology in figure 1.

$$\sum_{j \in J_i} (x_d^j + x_s^j) - x_d^i - x_s^i + x_{end}^i = a_i + x_{init}^i. \tag{4}$$

where x_{init} is the initial reservoir level, x_d is the discharge level, J_i is the space of hydro modules leading into module i, x_s is the spillage levels, x_{end} is the reservoir level at the end and a_i is inflow to reservoir i, which is a stochastic variable in the overall problem. However, for each decomposed LP-problem, a_i is known.

2.2.4 *Cuts*

The cuts used to create the FCF-approximation are represented as constraints in the LP-problem. These cuts also create a lower bound on the actual future cost. The more of these cuts we have the better approximation we get. A single cut takes the form

$$\alpha - \sum_i \phi_j^i x_{end}^i \geqslant b_j, \tag{5}$$

where α is the variable that represents the future cost in the objective function (see equation 2), ϕ_j^i is the slope for module i in cut j, and b_j the constant from cut j. See figures 3 and 4 for illustrations.

2.2.5 *Variable Limits*

We have physical limits associated with the hydro system. There is a minimum x^i_{min} and maximum x^i_{max} reservoir water levels and a maximum discharge x^{max}_d level within a week. There is a also a maximum capacity y^{max}_j for the thermal units. For all other variables the lower limit is 0 and the upper limit is ∞.

2.2.6 *Summary*

The complete LP-problem is thus
Minimize

$$J = \alpha + \sum_j c_j y_j \tag{6}$$

Subject to

$$\sum_i p_i x^i_d + \sum_j y_j = d \tag{7}$$

$$\sum_{j \in J_i} x^j_d - x^i_d - x^i_s + x^i_{end} = a + x^i_{init} \tag{8}$$

$$\alpha - \sum_i \phi^i_j x^i_{end} \geqslant b_i \tag{9}$$

$$x^i_{min} \leqslant x^i_{end} \leqslant x^i_{max} \tag{10}$$

$$x^{min}_d \leqslant x_d \leqslant x^{max}_d \tag{11}$$

$$0 \leqslant y_j \leqslant y^{max}_j \tag{12}$$

.

2.2.7 *Inflow Sampling*

Inflow Samples are in this project generated by using a single set of 30 inflow scenarios that spans over 52 weeks, and creating a normal distribution of this as in [7]. This may cause negative inflow-samples, which may lead to LP-problems that are impossible to solve. This is fixed by creating additional variables in the LP-problem, a variable that represents an external source of water, but when used gives a very high cost in the objective function, such that it is only used when no other option is viable.

The modified water balance for a module i then becomes

$$x^i_{init} + \sum_{j \in J_i} x^j_d - x^i_d - x^i_s + x^i_{end} + x^i_e = a, \tag{13}$$

where x^i_e is the amount of external energy bought for module i. To assure that this is only used when no other option is viable, this vari-

able is assigned a high cost in the objective function, which modified will take this form

$$J = \alpha + \sum_j c_j y_j + \sum_i M x_e^i, \tag{14}$$

where M has to be large enough so that $x_e^i > 0$ only when no other possible solution is viable.

2.3 INTRASTAGE TIME RESOLUTION

It is also of interest to use a lower time resolution *within* one problem. That is, instead of just dividing one week into seven days, creating seven different LP-problems, we can model that with a larger LP-problem. If we look at days instead of weeks for example, we get seven power balances (a daily demand that has to be met), and seven water balances for each hydro module.

2.4 PLANNING HORIZON WATER VALUE

At the end of the planning horizon we want some way to assert value to remaining water, so the optimal solution does not always cause empty reservoirs at the last stage. The longer the planning horizon, the less impact this approximation has to the optimal policy at first stag. In this paper this is done by first running the simulation with no value asserted to the water at the end of the planning horizon, then by looking at the marginal water values of that solution, we use the average of the marginal water value at the same day at the earlier years. This causes a good enough approximation for this work.

2.5 LP-RELAXATION

For each iteration the list of cuts grows proportionally with the number of iterations, which causes larger LP-problems which takes longer to solve. For increased efficiency, the actual LP-problem is initially solved with no cuts, then the cut which is most violated is added to the LP-problem and re-solved. This is done until no cuts are violated. A cut is said to be violated when $\alpha - \sum_i \phi_i^j x_{end}^i < b$, so by most broken we mean that given a solution $\{x_i\}$, the cut j where $\alpha - \sum_i \phi_i^j x_{end}^i - b$. has the lowest value.

2.6 CONVERGENCE

By running the forward iteration we get a proposed possible solution for each inflow scenario at each state, as illustrated in 6. As this solution is a viable solution, it represents an upper bound on the optimal

solution, as an optimal solution by definition is the minimum of all the possible viable solutions. This upper bound $J^+$is calculated by taking the sum of the average cost over all states at one stage (see 5). That is,

$$J^+ = \frac{1}{N_S} \sum_{t=1}^{T} \sum_{s=1}^{N_S} \sum_{j=1}^{N_M} c_j^{ts} y_j^{ts}. \tag{15}$$

As the approximation of the future cost function is a lower bound of the actual future cost, the value of the objective function at the first stage becomes a lower bound J^- of the total cost. That is,

$$J^- = \alpha_1 + \sum_{j} c_j^{0,0} y_j^{0,0}. \tag{16}$$

We say that the solutions has converged when the lower bound is *sufficiently* close to the upper bound. This is chosen to be, as in [12], to be when the lower bound is inside a 95% confidence interval of the upper bound. Because of the stochastic inflow, the upper bound calculated above is actually only an estimate of the upper bound. Thus we create a confidence interval around this estimate by using the estimated variance

$$s^2 = \frac{1}{N_S - 1} \sum_{t=1}^{T} \sum_{s=1}^{N_S} \sum_{j=1}^{N_M} (c_j^{ts} y_j^{ts} - J^+)^2. \tag{17}$$

A 95% confidence interval around J^+ is then given by

$$\left[J^+ - 1.965s, J^+ + 1.965s \right]. \tag{18}$$

Convergence is therefore said to be reached if lower bound J^- lies inside this interval.

Part III

PARALLEL SOLUTION

The parallel solution explained

3

PARALLEL SOLUTION

A parallel implementation means having several processors collaborate to solve a single problem. There are two major challenges when implementing an algorithm in parallel; 1) Being able to break down the algorithm into several parts, where the parts can run independent of each other (in parallel). 2) Minimizing the communication between processors since this has a significant cost attached to it.

Two important metrics in parallel computing is *efficiency* and *speedup*, which measures the quality of the parallel algorithm. Speedup S_P is defined as the factor which runtime has increased from the sequential execution, $\frac{p_1}{p_c}$, where p_c is the runtime with c cores. Efficiency is defined as the ratio between the number of processors N_P and the speedup, $\frac{N_P}{S_P}$.

As illustrated in figured 6 and 7, at each stage all the LP-problems are independent of each other. An obvious way to run this in parallel thus seems to be to calculate on stage at a time, sending one state to one processor, and advance to the next stage when this stage is done. In the forward cycle, one stage depends of the preceding stage, as the water level at the optimal solution of the preceding stage is used at initial reservoir levels at the current stage.

For the backward cycle, all the cuts created at one stage are used at a preceding stage. These cuts are used to approximate the future cost function. We can improve parallelism by simplifying this approximation to lessen the dependencies on stages calculated by other processors. This thus however mean that not all cuts are needed at all times from a mathematical point of view, it just comes at a cost of potentially worse approximation of the future cost function. One way could be to wait for all the cuts of the next stage, before starting of the previous stage. This does create a synchronization point, which leads to less efficient parallel solution. This mean that at each stage, all the processors have to wait for all the states at the next stage before starting on the previous stage.

We use a master processor to designate the decomposed LP problems to a set of slave processors. The forward iterations is performed along N_S forward samples (illustrated in figure 6). For each time stage in the backward iterations, the N_B backward realizations are considered for each of the N_S states obtained from the previous forward iteration. Thus, it is clear that the backward iteration is more computationally demanding, as it needs to solve N_B as many LP problems as in the forward iteration. In other parallel implementations of SDDP applied to the LTHS problem, there seems to be (at

least) two synchronizations points; between stages in both the forward and backward iterations. This has been reduced to 1 synchronization point in this work, only the forward is necessary.

3.1 FORWARD ITERATION

The forward iteration has to be completed before the backward-cycle starts, so we get a synchronization points in between the two cycles. Furthermore, within one inflow scenario the state at t used the solution at stage $t-1$ as its initial reservoir levels. Thus, all of the N_S LP problems formulated in each stage can be solved in parallel. However, due to the time-sequential coupling along scenario samples in the forward iteration, one cannot expect speedup in the forward iteration if $N_P > N_S$. The designation of LP problems in the forward cycle is illustrated in figure 8.

3.2 BACKWARD ITERATION

For each evaluated state in a stage t in the backward iteration, a cut created for stage $t-1$ by averaging contributions from the LP problems solved corresponds to the N_B realizations of inflow. Each new inflow realization will in practice introduce a modest change in the LP-problems right-hand side. Thus, the LP problem can normally be solved within a relatively low number of simplex iterations, if the previous solution basis is available. In this work, the advantage of warm-starting LP-problems in the backward iteration was appreciated by letting a designated processor solve all N_B problems origination from a given initial state. Allowing the N_B samples from a given state to be divided between different processors could add flexibility to the parallel processing scheme, but one would lose some of the warm-start advantage. We have focused on limiting the communication between the processors, and thus, the communication of warm start basis was not considered. Due to the linearity of the model, a cut created from sample s_1 in stage $t-1$ is valid for all states in that stage. Although convenient from an implementation point of view, it is not mathematically necessary to wait for all processors to create a cut for stage $t-1$ before continuing backwards in time to construct cuts for stage $t-2$. Each additional cut considered for stage t gives a better approximation to the future cost function. However, waiting for all cuts to be created forces all processors to be synchronized at each stage in the backward iteration. In presented work, this stage-wise synchronization point in the backward iteration is relaxed, allowing each processors to wait for N_W cuts, where $N_W \leqslant N_S$. By setting $N_W = 1$ we fully relaxed the stage-wise synchronization point, and no processors is unused at any time during the backward cycle. By setting $N_W = N_S$ we have full synchronization earlier. For $1 < N_W < N_S$

we have partial synchronization. The cost of not waiting for all cuts may be slower convergence, as in more iterations needed before convergence is reached. Different values of N_W has been tested in this paper. The designation of LP problems from master to slave is illustrated in figure 9 for $N_W = 1$ and 10 for $N_W = N_S$.

The synchronization points after each stage are removed. This way processor time are utilized more efficiently, as each processor does not have to wait for others to have finished before proceeding. The cost is that having less cuts might mean a reduced approximation to the future cost function, which in turn might lead to more iterations needed before having reached convergence. The question is whether using processors more efficiently outweigh the negative effect from having a lower amount of cuts at each stage. All other parts of the algorithm is described as above.

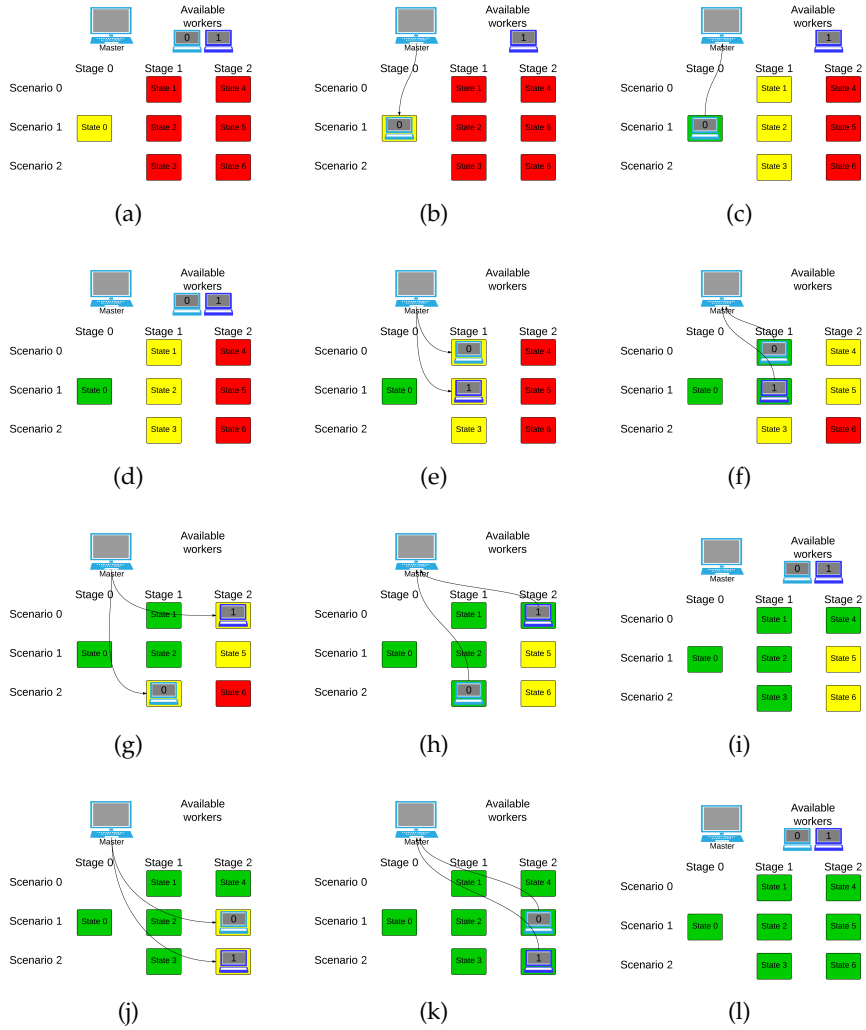
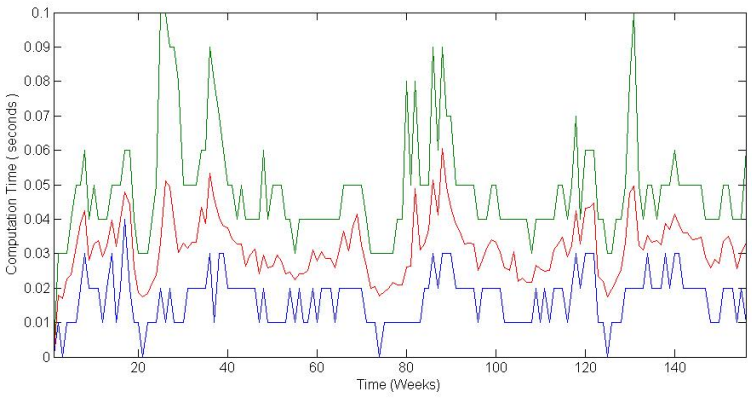


Figure 8: Parallel forward cycle delegation.



3.3 COMMUNICATION

The exact amount of data transferred between processors is explained here.

3.3.1 *Slave to Master*

In the forward cycle, the slave has to send the reservoir solutions to the master to be used as initial reservoir for the next stage. It also has to send the current cost as described in equation 2 which is used to calculate the upper bound, as described in chapter 2. This is $N_M + 1$ numbers, as each hydro module has a reservoir, and the current cost is a single data.

In the backward cycle, the slave sends the new cut it has generated, which consists of $N_M + 1$ data, as described in chapter 2.

3.3.2 *Master to Slave*

From master to slave, all *new* cuts that the master has received for the stage since the last time it sent data to that slave for that stage. A slave locally stores all cuts it has ever received. This amount will vary, but on average it will almost equal the number of inflow scenarios N_S *per iteration per stage*, as all cuts created will sooner or later be transferred to the slave, except for the last iteration. It is worth noting that this is the only part of the algorithm where communication actually increases with the number of processors.

In the forward cycle, the master also has to send the initial reservoir to the slave, which is N_M data.

So the average total amount of data from master to all slaves in one iteration is $N_S(N_M + 1)TNSl$, since we will have $(N_S N_M)$ cuts created for each stage, T stages, which will be sent to N_{Sl} number of slaves.

3.4 HARDWARE AND LIBRARIES

All results have been obtained on the Linux cluster Kongull which consists of 93 compute-nodes, each equipped with 2x 6-core 2.4 GHz AMD processors and 24 GiB The cluster operating system is CentOS 5.4. For more details see [2]. To solve LP-problems an open source simplex solver from the Coin-CLP library [1] was used. The solver uses the dual-simplex algorithm. For parallel communication MPI-library OpenMPI has been used.

3.5 MPI

For message passing between processors MPI was used. MPI has several ways to transfer messages between processors. For this algorithm it is important that the processors does not have to wait for a matching receive post to continue what they are doing. That is, we want a processor to initiate a send without having to wait for the master processor to actually be ready to receive before continuing doing other stuff. MPI_Isend serves this purpose, while the standard receive function MPI_recv was used for receiving messages. The function MPI_Wait were used where it was necessary to make sure messages were not overwritten before they were sent. MPI_Wait, MPI_Probe and MPI_Iprobe were used to wait for a message or test if there are pending messages to be received.

3.6 DETAILS OF IMPLEMENTATION ON MESSAGE PASSING

There were some key challenges of the implementation of message passing.

It was important to minimize the idle times by implementing a way to trigger a send post without having to wait for a matching receive post before continuing to do calculations. For example, the master processor has to be able to trigger a send post to a slave that is *not yet* ready to receive that message, without having to wait for the slave to be ready to receive the message before continuing. We want a slave to receive a new problem *immediately* after finishing the previous problem. We also want a slave to be able to start on a new problem before a master necessarily have received the solution to the old problem. The MPI library has a built in support for this, which is called *nonblocking message passing* (MPI_Isend and MPI_Irecv).

Another key challenge is the selection of which slave to send the next problem. Even though we want to be able to delegate new problems to slaves that are currently busy, we do not want to delegate *more than one* problem to an already busy slave, as we want to send a problem to the slave that is most likely to be ready to solve that problem first, which will be more uncertain the more jobs that are already

in queue. The function MPI_Waitany from the standard MPI library should in theory work quite well for this purpose, but a slower than expected performance was experienced (which probably has something to do with buffered messages). This was solved by creating a FIFO queue of slaves, which was initiated with two instances of every slave. Every time the master delegates a new problem, it's delegated to the first in queue. Every time the master receives a message from a slave, it is added back into the queue. This asserts that a slave never has more than two jobs assigned.

In summary, from a slave point of view, the implementation is:
1.) Receive a message with MPI_Recv.
2.) Solve delegated problem.
3.) Send new message with MPI_Isend.

From the master point of view it is the following,

while There is a problem to be solved **do** Delegate a problem to first slave in queue with MPI_Isend
    if A slave has sent a message **then**
        Receive message with MPI_Recv and add slave to queue.
    end if
end while

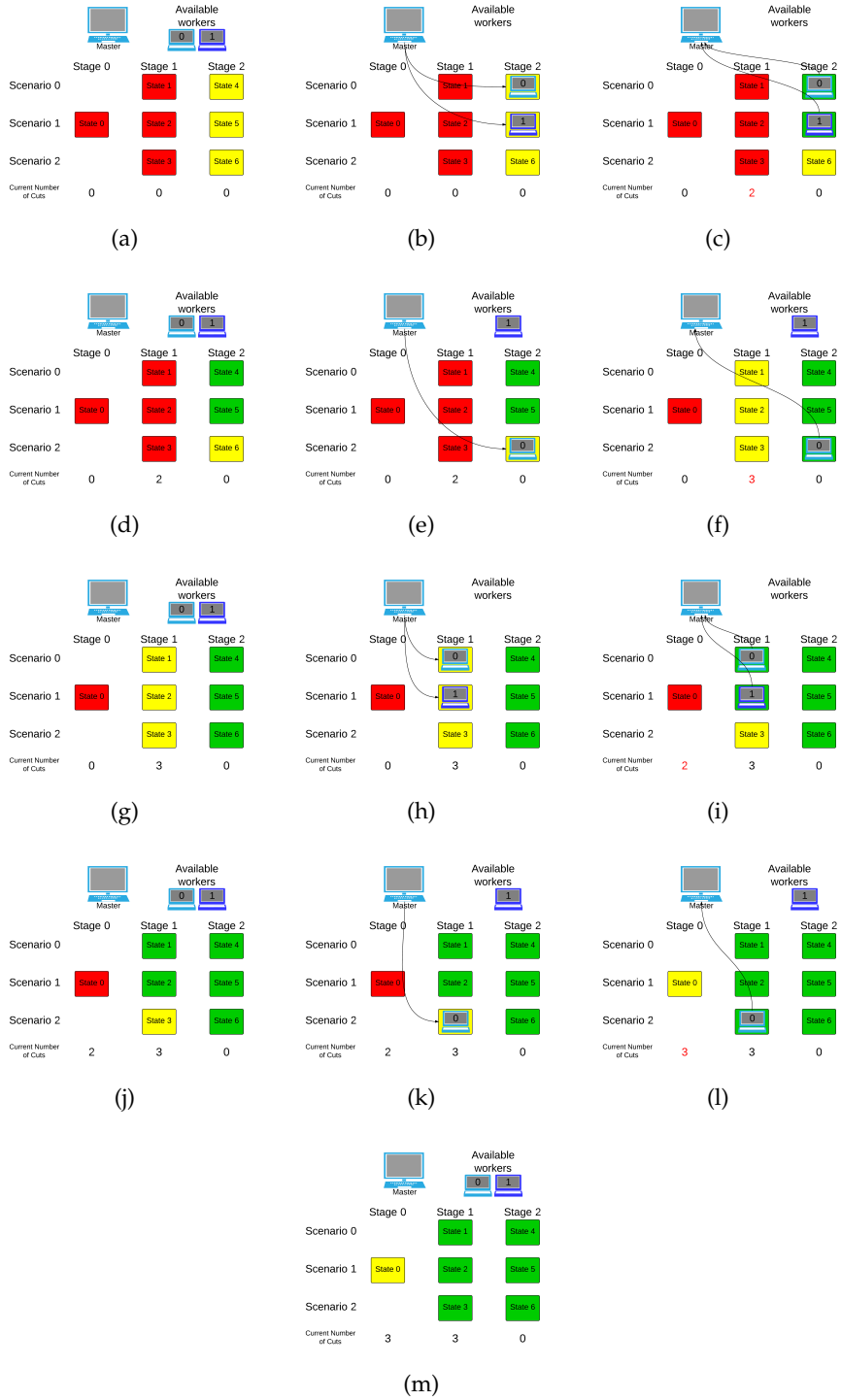


Figure 9: Parallel backward cycle delegation.

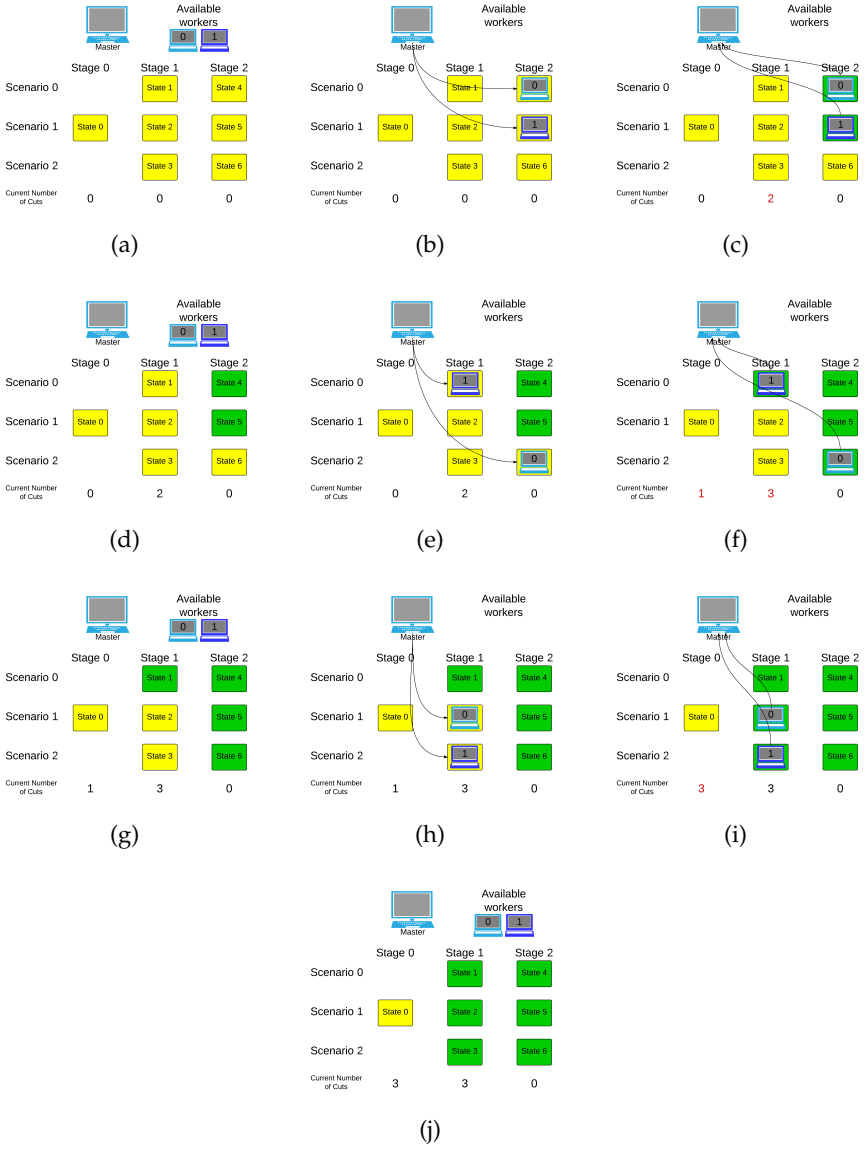


Figure 10: Parallel backward cycle delegation.

Part IV

VERIFICATION OF CORRECTNESS

The parallelization breaks the deterministic nature of the algorithm, because which and how many cuts evaluated at any given state is random (as a result of small fluctuations in computing times), the process of verifying that the parallelization works correctly is non-trivial. The way it is done here is by presentation of various results given by parallelized algorithm and showing that it makes sense.

4 PRESENTATION OF SOLUTION

4.1 HYDRO TOPOLOGY

The hydro system investigated was the one in Nea-Nidelva, which consists of 12 hydro modules. The hydro topology is shown in figure 11 and the modelling of individual hydro modules are illustrated in Figure 12.

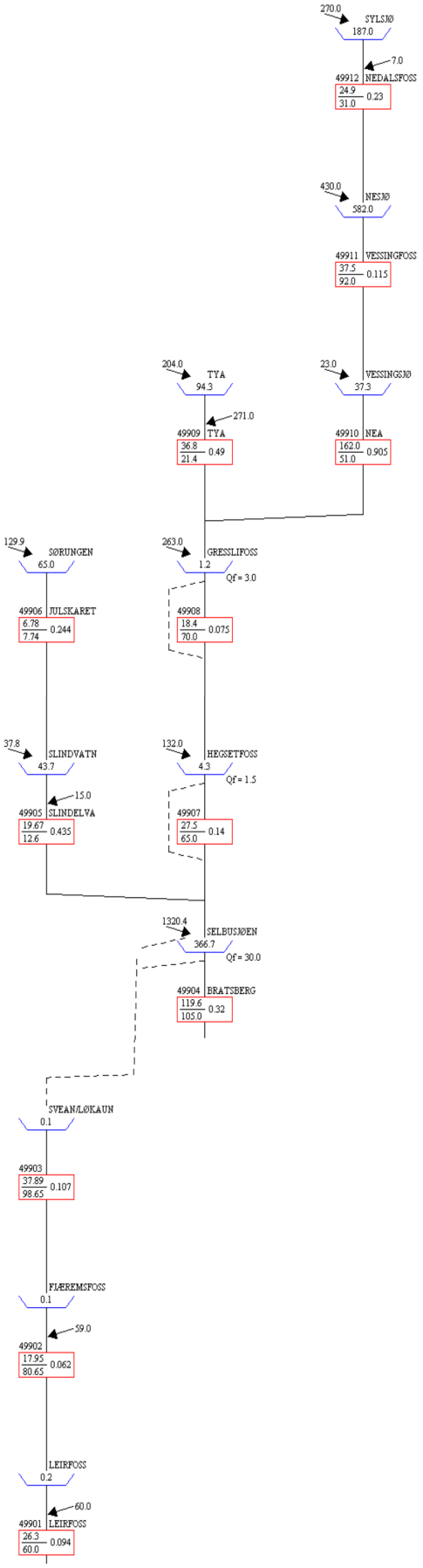


Figure 11: Hydro topology.

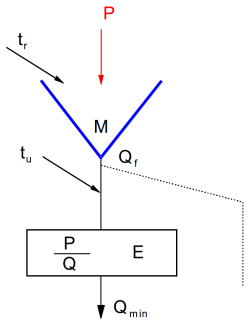


Figure 12: Hydro module illustration, see table 1 for more information.

M	Reservoir capacity (Mm^3)
P	Max power (MW)
Q	Max discharge (m^3/s)
E	Energy equivalence (kWh/m^3)
t_r	Yearly average inflow (Mm^3/year)

Table 1: Hydro module description. Only values used in this paper are explained.

4.2 INFLOW, DEMAND AND THERMAL COST PROFILE

The inflow data that has been used to generate inflow samples can be seen in figure 13, and the inflow samples generated can be seen in figure 14. As can be seen, the variance is somewhat lower in the generated samples than in the original data.

Figure 16 shows the demand for energy (load) and how it varies throughout the year. Figure 15 shows the marginal cost profile for thermal energy. Both the load and thermal cost profile are chosen rather arbitrarily, but chosen such that the total yearly demand exceeds the energy produced from the hydro system alone to avoid a trivial solution.

4.3 VERIFICATION OF CORRECTNESS

Usually, verification of the parallel implementation being correct consists of verifying that it gives the same exact result as the sequential implementation. However, this parallel implementation has some non-deterministic aspects which are impossible to reproduce. The amount of cuts used by a processor at one stage can differ since the time spent by one processor at one state is not deterministic, the actual outcome can change. Thus the verification done here is by illustrating various results and see that they look sensible. Another

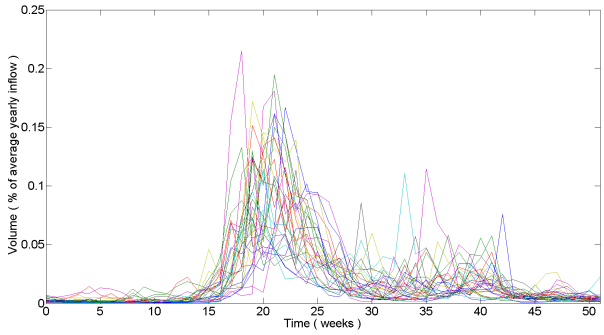


Figure 13: Inflow data used to generate inflow samples.

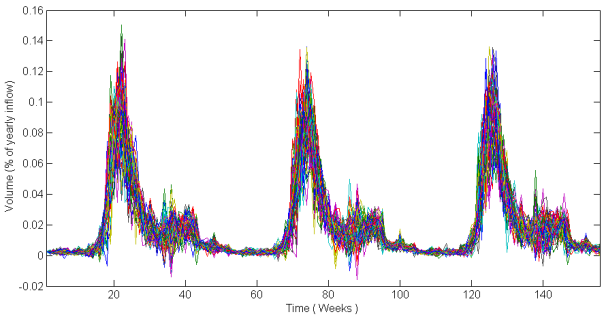


Figure 14: Generated inflow samples

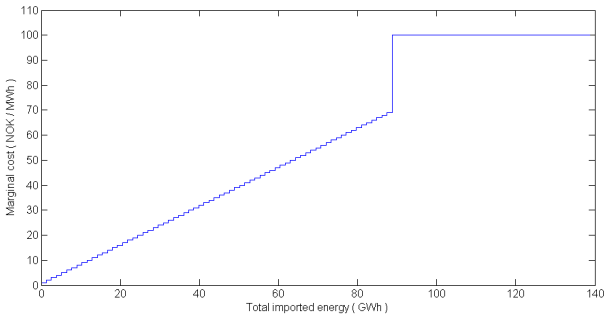


Figure 15: Cost profile of thermal energy.

thing worth mentioning is that a sequential implementation for this algorithm was never made. Sequential here means running one two processors, one acting as master. But for all practical purposes this would equal the algorithm run at a single a processor.

4.4 SETTINGS

All solutions presented in this section is acquired with the following settings. - 12 Processors (11 slaves)
- Planning horizon of 3 years where one stage is one week, in total 156 stages

- 7 intraweek stages (days)
- Full synchronization
- Generated inflow samples as presented in Figure 14
- Hydro topology as seen in Figure 11
- Thermal cost profile as seen in Figure 15
- Demand profile as seen in Figure 16
- Initial reservoir set to 60% of reservoir capacity

Figure 17 shows the reservoirs levels for the final solution. As can be seen, the reservoir levels are high in the season where inflow is high, and then is gradually emptied such that it is almost empty right before the inflow becomes high again the next year. This is what to be expected in a good solution.

Figure 18 shows the usage of external water sources (as described in Chapter 2).

Figure 19 shows the total energy produced at each stage, and in Figure 20 it's illustrated together with the demand profile. As expected, the production follows the demand curve quite closely.

Every week a demand of energy has to be met, either by producing it from the hydro system or by buying thermal energy. The marginal thermal energy prices increase with the amount bought, so a tendency of a good solution should be that the amount of thermal energy bought should be relatively stable. Because of the increasing marginal prices, if a large amount is bought one week, and a small amount another week, there would probably have been better to store more of the water in the the first week to be used in the latter. As can be seen in Figure 21 the solution seem to follow this structure. The oddities that can be found can also arises at spring flow when the amount of water exceeds the capacity so that storing more water would increase risk of spillage such that expected profit may suffer.

A similar aspect should be that the marginal value of the water at one module (which is the shadow price of the reservoir variable in the LP-problem), should be relatively constant over the whole period. As can be seen in Figure 23 and 22, this seems ok. The different modules differ in marginal values both because they differ in efficiency, but also because of their relative position to each other. A module upstream of another module must have a higher marginal water value.

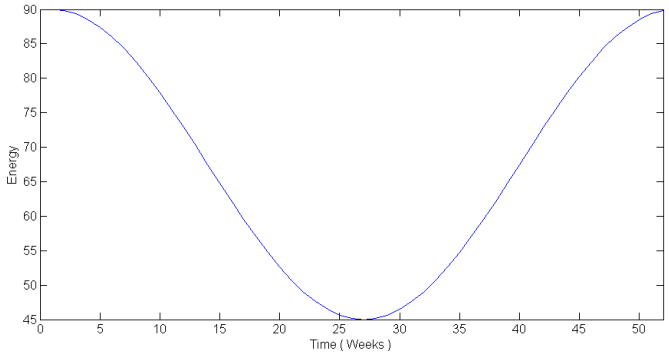
As can be seen in figure 23, the 0 and 100 percentile varies somewhat at the end of the planning horizon. This is caused by the incomplete assertion of water values at the end of the planning horizon, as described in chapter 2. The system seems reasonably well balanced, have a rather flat power price (marginal cost of power) and water values.

4.5 VERIFICATION OF ALGORITHM EFFICIENCY

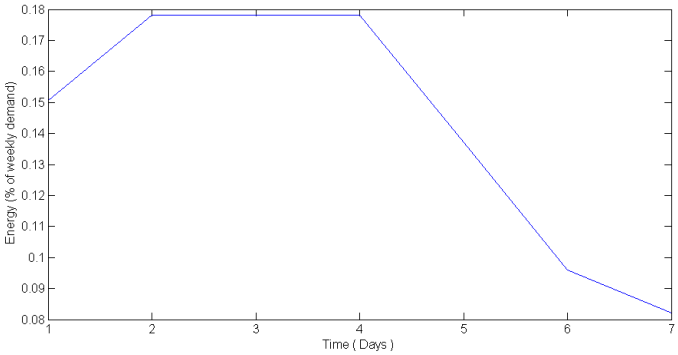
Some testing of the algorithm efficiency has been done, and compared to known existing algorithms.

Figure 24 shows exactly how much of the computational time that is spent solving the LP-problems. This is known to be at 80-90% for implementations used in the industry today. Since this implementation is somewhat less efficient we might get somewhat better parallel results than can be obtained in a more efficient implementation.

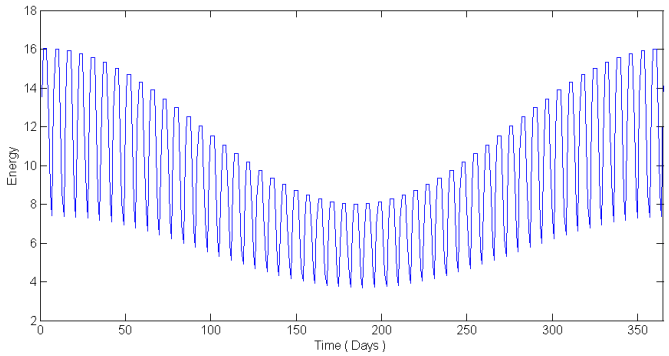
Figure 25 shows the amount of time spent in the forward cycle vs the time spent in the backward cycle. As 12 LP problems are solved for each state in the backward iteration, and only one LP problem for each state in the forward iteration, one could in theory see more than 90% of the time used in the backward iteration. The main factor that this is not the case is most likely because of the warm-starting of the LP-problems that is exploited in the backward iteration.



(a) Weekly demand profile.



(b) Daily demand profile.



(c) Complete demand profile for one year.

Figure 16: Demand profiles.

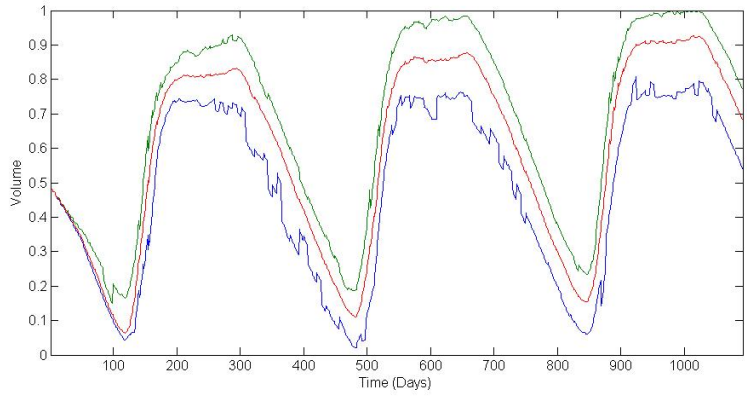


Figure 17: Reservoir solutions, average, 0 and 100 percentile.

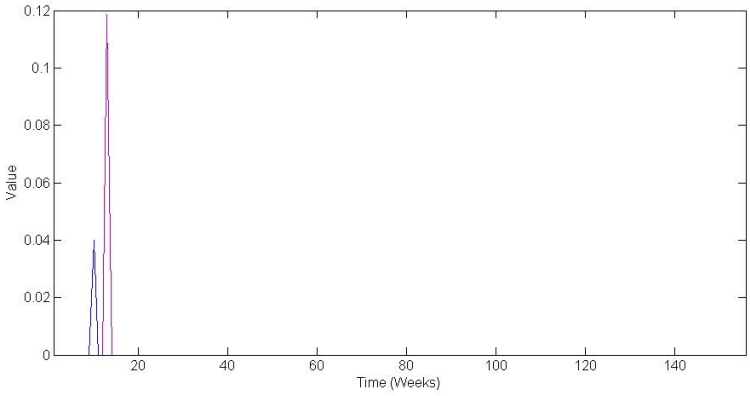


Figure 18: Usage of external water sources.

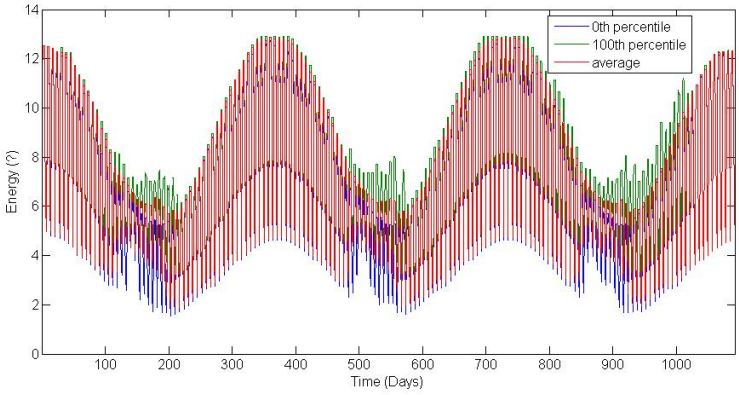


Figure 19: Total energy produced.

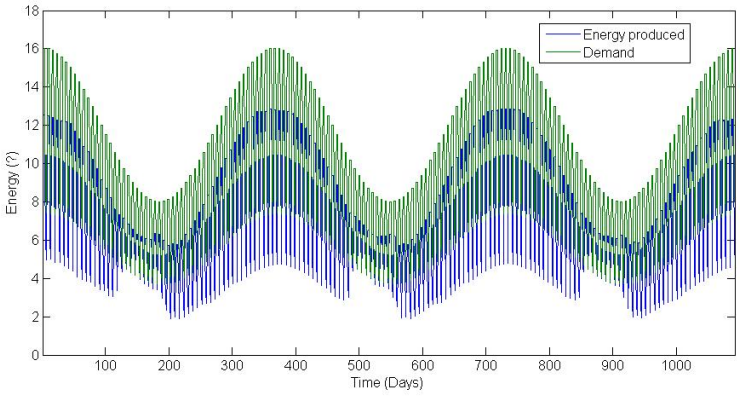


Figure 20: Total energy produced vs demand.

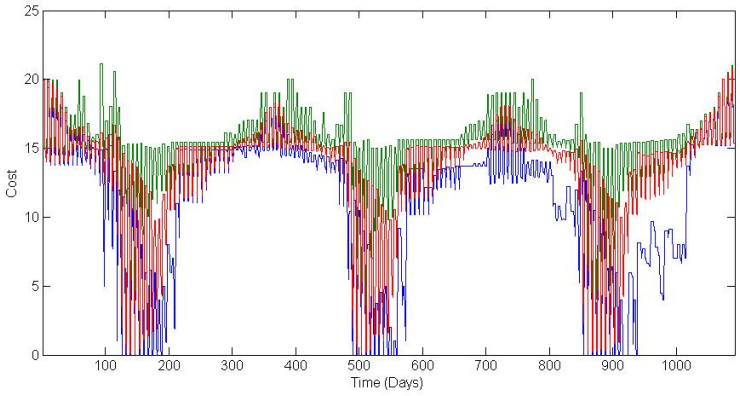


Figure 21: Average, 0 and 100 percentile of energy marginal cost.

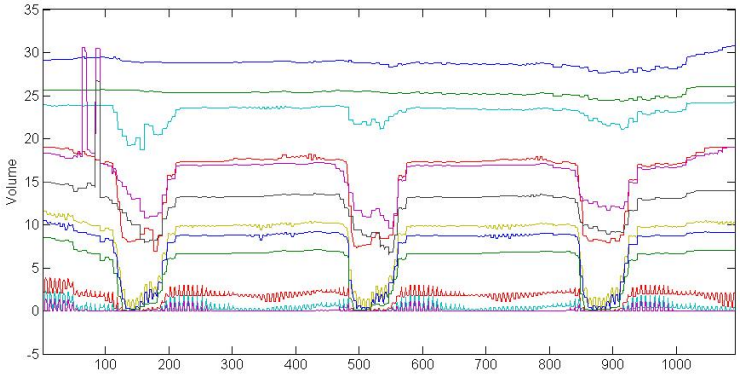


Figure 22: Average marginal water values for the different modules.

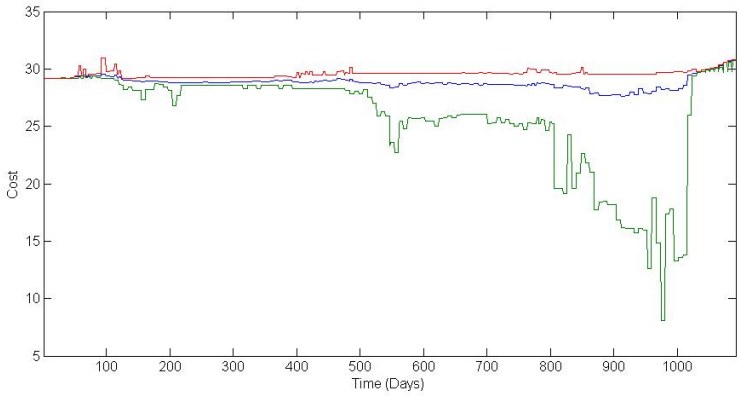


Figure 23: Average, 0 and 100 percentile for marginal water value of a single module.

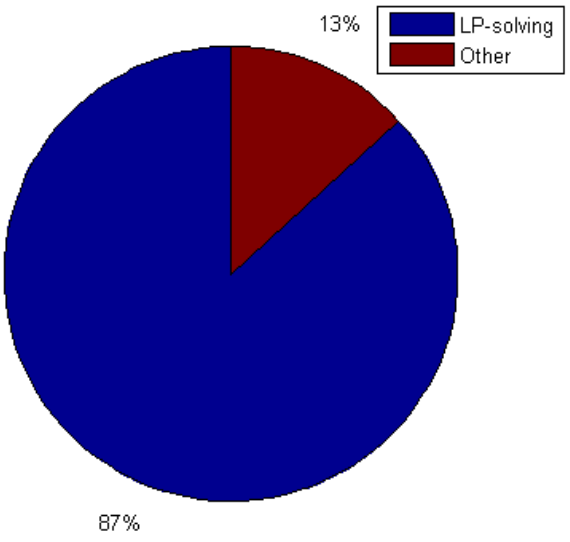


Figure 24: Time spent LP-solving.

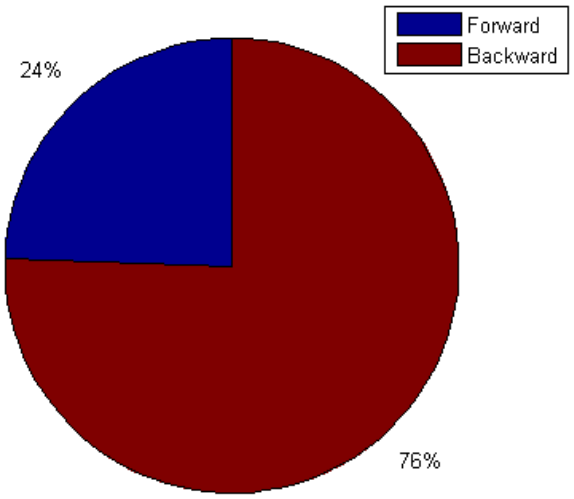


Figure 25: Proportion of time spent in forward and backward cycle.

Part V

RESULTS

This part contains the results of two case studies with different values of the number inflow scenarios and backward samples.

5 RESULTS

Two case studies have been done with different values of inflow scenarios N_S and backward samples N_B, as shown in Table 2. These settings are similar to those being used in many operational models. The maximum number of processors deliberately does not exceed N_S as there is not implemented a way to calculate one state on two processors, such that any amount of processors above N_S will not be very effective in the forward iteration. For both cases we have experimented with different numbers of processors and the number of cuts to wait for N_W at each stage.

The optimal speedup is usually equal to the number of cores that its run on. On c processors the best one can usually hope for is a program that run a factor of c faster. However, in this particular problem, one can in theory see a "super optimal" speedup. When relaxing the synchronization points one LP problem at a particular state could have less cuts, which means potentially faster solution times. If the cuts abandoned would not be active cuts, such that convergence is reached in the same amount of iterations we could see super optimal speedup. This can be seen in for the simulations with few processors and relaxed synchronization points. By relaxing the synchronization points, one gets less idle time for the processors, but might get slower convergence. As these simulations have been done with a master-slave setup, speedup and efficiency here has been measured as a function of the number of slaves, not the number of processors. Sequential runtime is here defined as the one run on two processors as no separate sequential implementation has be done. In the parallel algorithm there are typically two bottlenecks, communication between the processors, and synchronization. With synchronization points, one forces the processors to be at the same place in the algorithm at the same time, and since the problems delegated (that is, solving the LP-problems) can vary in solution times, this may cause processors to wait for each other.

As seen in Figure 27 and 26, with full synchronization the average idle time spent for a slave is as high as 45 percent, while the average time spent communicating is as low as 3 percent. This means that it is

	N_S	N_B	Max no. processors
Case 1	71	12	72
Case 2	200	50	144

Table 2: The two cases that has been simulated.

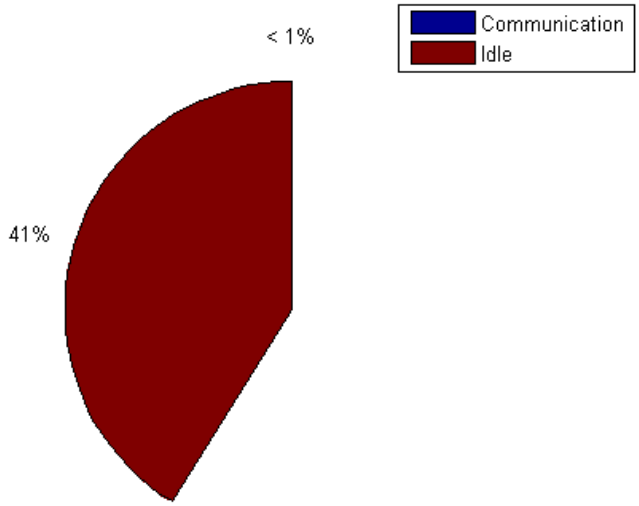


Figure 26: Average time spent on communication and idle time for a slave on case 2 with 144 processors and $N_{min} = 200$.

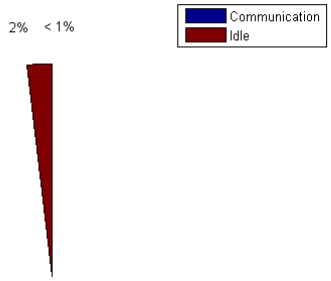


Figure 27: Average time spent on communication and idle time for a slave on case 2 with 144 processors and $N_{min} = 1$.

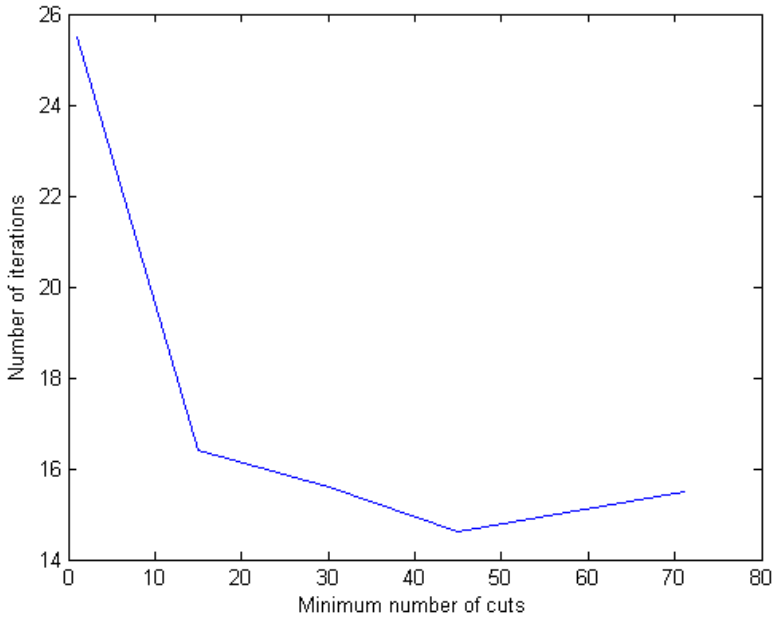


Figure 28: Number of iterations until convergence is reached for case 1 with 72 processors.

the synchronization that is the bottleneck for this implementation. On the other hand, by completely relaxing the synchronization points, we see in Figure 27 and 26 that the idle time is reduced from 45 percent to 3 percent. The downside is that more iterations are needed to reach convergence, as can be seen in Figure 28 and 29.

this is because on average fewer new cuts are used, as seen in Figure 37. By partial synchronization we also see in Figure 37 that the average number of cuts are drastically increased while the idle time is only slightly increased. This causes for the best solution times.

As seen in figures 31 and 33, removing the synchronization points is very effective until the number of processors get too high relative to the number of forward inflow scenarios. At some point the number of cuts at each iteration is so low that it nearly doubles the iterations needed before convergence, which results in severely lower speedup. This thus however lead to the thought that partially relaxing the synchronization points, which specifically means that wee force each processors to wait for a specific amount of cuts between 1 and the number of inflow scenarios N_S, might be optimal. The idea is that waiting for 2 cuts instead of 1 cut will barely affect computational time at all, while having a relatively large chance of affecting number of iterations needed, while waiting for the 60th cut instead of the 59th cut will have a lower chance of affecting number of iterations needed. In addition, as seen in Figure 30, there is a variation in time spent solving different LP problems. And with full synchronization, when one processor spends more time solving one LP-problem, all

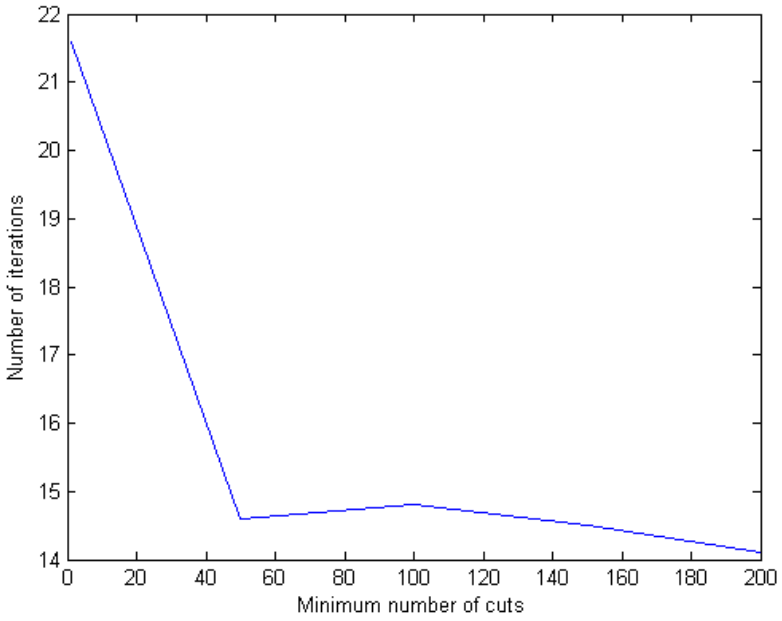


Figure 29: Number of iterations until convergence is reached for case 2 with 144 processors.

the other processor have to wait for that one to finish. So trying to force the slaves to wait for a number in between might be the optimal solution, which is confirmed in Figures 32 and 34.

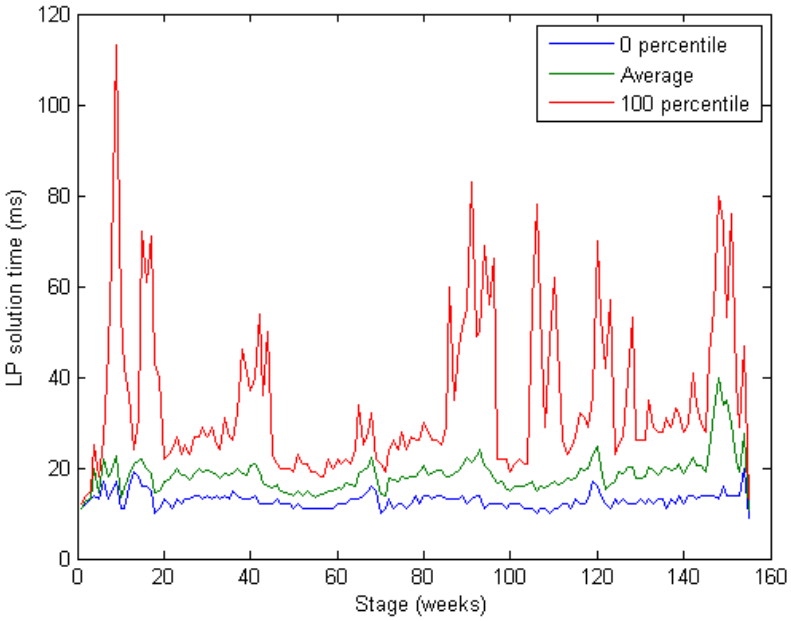


Figure 30: Average, 0 and 100 percentile for solution times in the backward iteration.

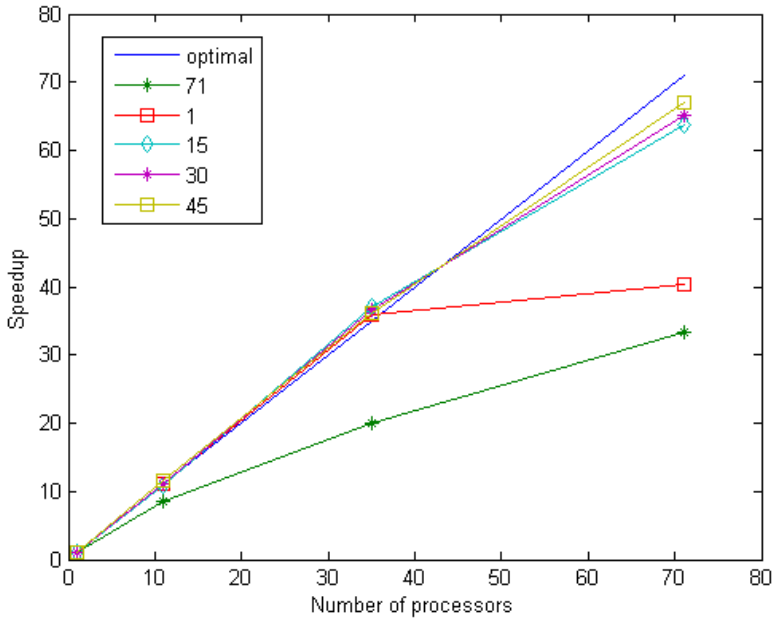


Figure 31: Speedup for case 1.

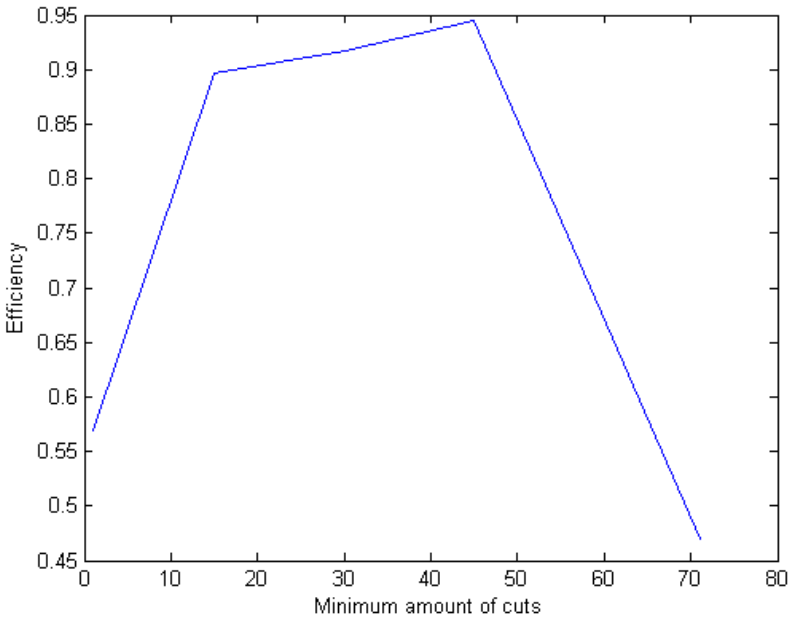


Figure 32: Efficiency for case 1 with 72 processors.

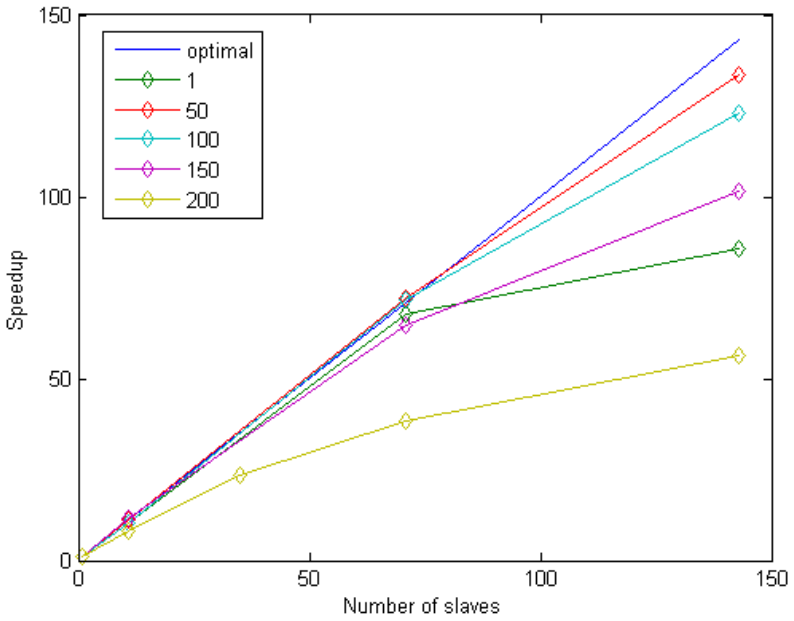


Figure 33: Speedup for case 2.

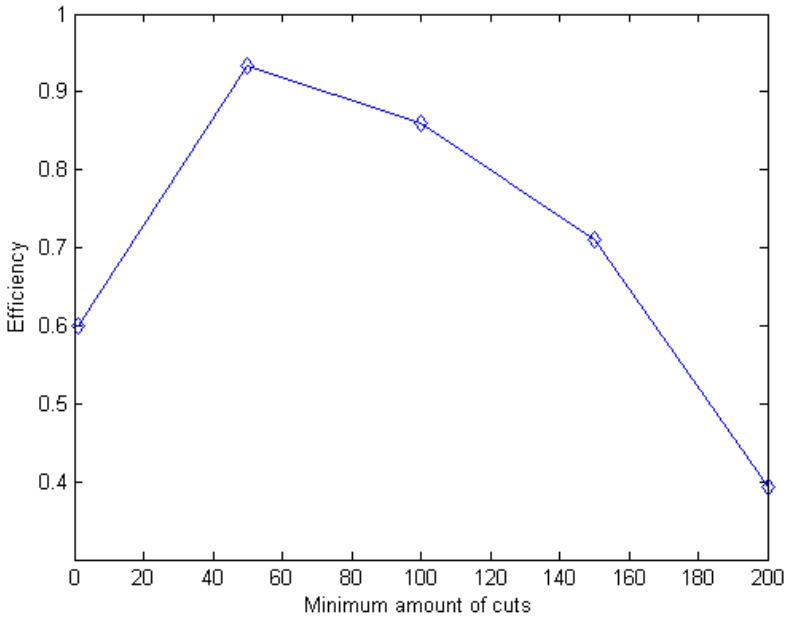
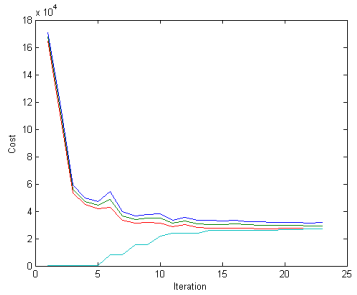
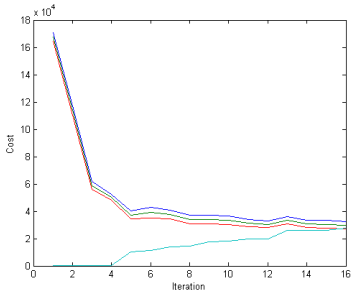


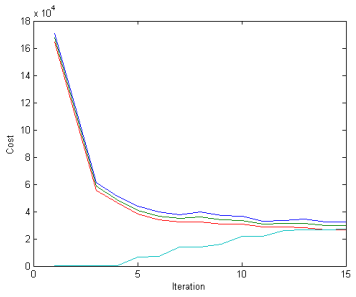
Figure 34: Efficiency for case 2 with 144 processors.



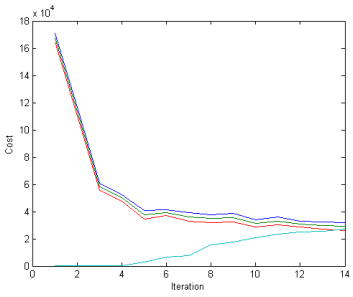
(a) $n_{min} = 1$.



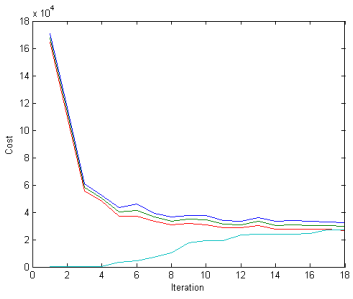
(b) $n_{min} = 15$.



(c) $n_{min} = 30$.

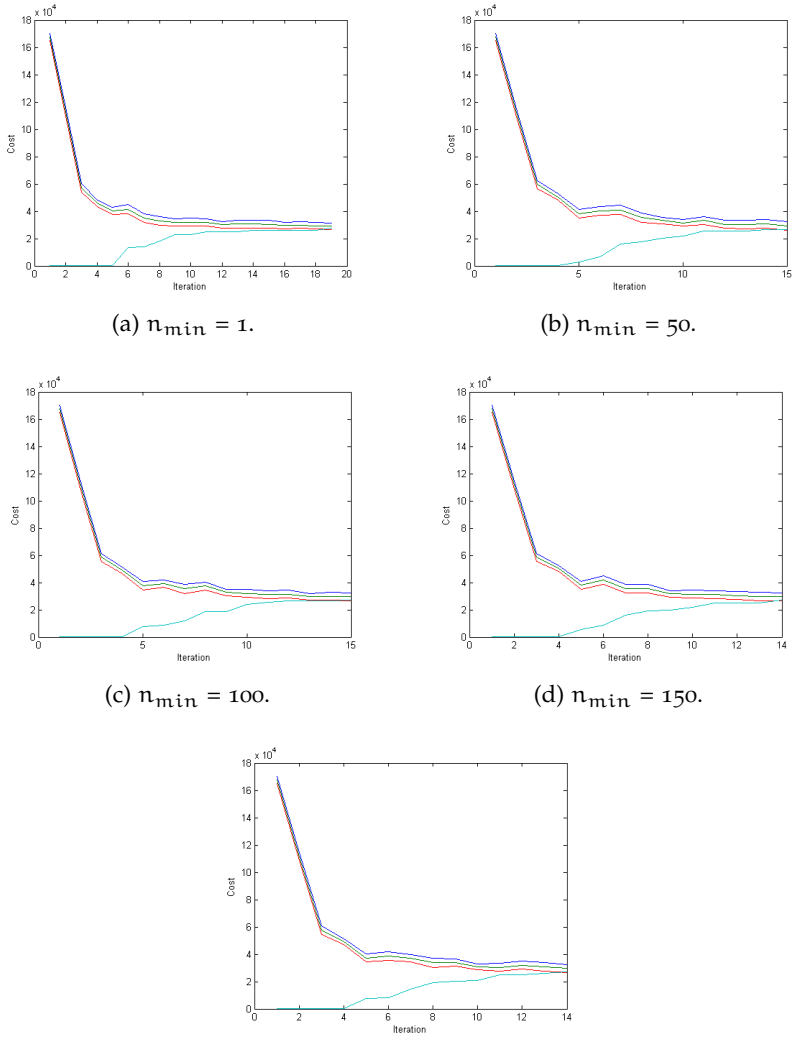


(d) $n_{min} = 45$.



(e) $n_{min} = 71$.

Figure 35: Convergence for different values of n_{min} in case 1.



(a) $n_{min} = 1$.

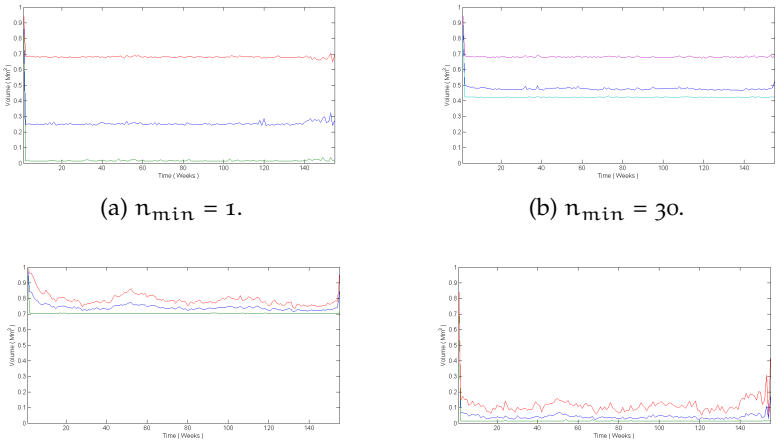
(b) $n_{min} = 50$.

(c) $n_{min} = 100$.

(d) $n_{min} = 150$.

(e) $n_{min} = 200$.

Figure 36: Convergence for different values of n_{min} in case 2.



(a) $n_{min} = 1$.

(b) $n_{min} = 30$.

(c) $n_{min} = 50$.

(d) $n_{min} = 1$.

Figure 37: Average number of cuts received where n_{min} has been varied.

Part VI

CONCLUSION

6 CONCLUSION

A parallel scheme for the SDDP algorithm used in long term hydro power scheduling has been proposed. Where earlier proposed parallel schemes have included synchronization points at each stage in the backward iteration, this paper challenges this by either partially or fully removing the synchronization points, at the cost of possible slower convergence measured in the number of iterations needed before convergence is reached. Two case studies have been done with different values for the number of inflow scenarios and the number of inflow samples in the backward iteration. The case studies have been done on a realistic model of a Norwegian water course. The results in both case studies indicate a significant increase in parallel efficiency by removing the synchronization points, especially in some of the cases done with partial synchronization, where in the most extreme cases the efficiency have more than doubled.

The case study reflect a simplified version of operational data, both in terms of system size and physical details being modeled. However, we believe that the presented case study results demonstrate a significant potential for improvement in parallel efficiency of operational SDDP models.

BIBLIOGRAPHY

[1] Coin-clp. https://projects.coin-or.org/Clp. Accessed: 2014-05-03.

[2] Kongull hardware. https://www.hpc.ntnu.no/display/hpc/Kongull+Hardware. Accessed: 2014-05-03.

[3] L. A. Barosso. Distributed processing in stochastic multi-stage hydroscheduling. *In In proc. of Hydro Scheduling in Competive Electricity Markets*, 2008.

[4] V. L. de Matos and E. C. Finardi. A computational study of a stochastic optimization model for long term hydrothermal scheduling. *International Journal of Electrical Power and Energy Systems*, 43:1443–1452, 2012.

[5] A. Gjelsvik, M. Belsnes, and A. Haugstad. An algorithm for stochastic medium-term hydrothermal scheduling under spot price uncertainty. *Power System Computation Conference*, 1999.

[6] A. Gjelsvik, B. Mo, and A. Haugstad. An algorithm for stochastic medium-term hydrothermal scheduling under spot price uncertainty. *Power System Computation Conference*, 1999.

[7] Anders Gjelsvik. Stokastisk tilsigsmodell for driftsplanlegging. 1992.

[8] A. Helseth, B. Mo, and G. Warland. Long-term scheduling of hydro-thermal power systems using scenario fans. 2010.

[9] G. Infanger and D. P. Morton. Cut sharing for multistage stochastic linear programs with interstage dependency. *Mathematical Programming*, 75:241–256, 1996.

[10] J. W. Labadie. Optimal operation of multireservoir systems: State-of-the-art review. *Journal of Water Resources Planning and Management*, 130:93–111, 2004.

[11] O.Wolfgang, A. Haugstad, B. Mo, A. Gjelsvik, I. Wangensteen, and G. Doorman. Hydro reservoir handling in norway before and after deregulation. *Energy*, 34:1642–1651, 2009.

[12] M. Pereira, N. Campodonico, and R. Kelman. Application of stochastic dual dp and extensions to hydrothermal scheduling. 1999.

[13] M. V. F. Pereira. Optimal stochastic operations scheduling of large hydroelectric system. *Electrical Power and Energy Systems*, 11(3):161–169, 1989.

[14] M. V. F. Pereira and L. M. V. G. Pinto. Stochastic optimization of a multireservoir hydroelectric system: A decomposition approach. *Water Resources Research*, 21(6):779–792, 1985.

[15] Roberto J. Pinto, Carmen L. T. Borges, and Marie E.P Maceira. An efficient parallel algorithm for large scale hydrothermal system operation planning. 2013.

[16] S. Stage and Y. Larsson. Incremental cost of water power. *Trans. Am. Inst. Electr. Eng.*, 80:461–364, 1961.

[17] A. Turgeon and R. Charbonneau. An aggregation-disaggregation approach to long-term reservoir management. *Water Resources Research*, 34:3585–3594, 1998.