The Journal of Supercomputing (2019) 75:1580–1593
https://doi.org/10.1007/s11227-018-2692-z

A CUDA approach to compute perishable inventory control policies using value iteration

G. Ortega[1] · E. M. T. Hendrix[1,2] · I. García[1]

Published online: 16 November 2018

Abstract
Dynamic programming (DP) approaches, in particular value iteration, is often seen as a method to derive optimal policies in inventory management. The challenge in this approach is to deal with an increasing state space when handling realistic problems. As a large part of world food production is thrown out due to its perishable character, a motivation exists to have a good look at order policies in retail. Recently, investigation has been introduced to consider substitution of one product by another, when one is out of stock. Taking this tendency into account in a policy requires an increasing state space. Therefore, we investigate the potential of using GPU platforms in order to derive optimal policies when the number of products taken into account simultaneously is increasing. First results show the potential of the GPU approach to accelerate computation in value iteration for DP.

Keywords GPU · Inventory control · Value iteration · CUDA

1 Introduction

As inventory control is a dynamic process also dynamic programming has been considered an appropriate technique to derive so-called order policies, see [15]. When

This research is partly funded by Project TIN2015-66680 financed by the Spanish Ministry and Spanish network CAPAP-H6 (TIN2016-81840-REDT). G. Ortega is a fellow of the Spanish "Juan de la Cierva Incorporación" program (Grant No. IJCI-2016-30173).

✉ E. M. T. Hendrix
eligius.hendrix@wur.nl; eligius@uma.es

G. Ortega
gloriaortega@uma.es

I. García
igarciaf@uma.es

1 Group of Supercomputation-Algorithms, Computer Architecture, Universidad de Málaga, Málaga, Spain

2 Operations Research and Logistics, Wageningen University, Wageningen, Netherlands

considering for a perishable good the complete age distribution as state variable, the state space increases drastically [7] rendering a challenge from a computational point of view. Obtaining good order policies for perishable goods is of great importance as about one third of world food production is wasted according to the FAO in 2011. Basically, if one orders too much, the tendency is high that products perish and waste is generated. When one orders too few, one may incur lost sales, i.e., having empty shelves during part of the day. An extreme case of one day perishability is the newsvendor problem, where one maximizes expected profit with a focus on balancing out-of-stock (OOS) and waste. Let s be the sales price of the product, c its unit cost and G the cumulative distribution function (cdf) of demand; then, it is known in literature that the optimal order quantity is due to the critical fractile formula

$$Q = G^{-1}\left(\frac{s-c}{c}\right). \tag{1}$$

Haijema and Minner [6] extend this problem to several periods of shelf life and investigate several order policies. They claim that an optimal dynamic programming (DP) policy has many advantages in this case.

However, in many cases, a so-called *stock base* policy works well, where an order-up-to level S is used, [11]. The retailer inspects the current inventory level I and orders an amount Q up to level S, i.e., the rule $Q = (S - I)^+$ is followed, where $(x)^+ = \max\{x, 0\}$. Due to modern scanning techniques, the retailer may be aware of the exact age distribution of the items in stock, let say $(I_1, \ldots I_r, \ldots, I_M)$, where the products with remaining shelf life in I_1 will perish at the end of the day, but fresher products (I_M is the number of items with the maximum shelf life M) can still be sold in the future. For some inventory models, it is possible to derive the so-called optimal order policy $Q(I_1, \ldots, I_M)$ via Stochastic Dynamic Programming (SDP) using a computational procedure which is called value iteration (VI).

Recently, an interesting question has been raised where two products interact with each other due to substitution of one product by the other when one is out of stock, [2,4,5,16]. It is much harder to obtain an optimal policy, as the order quantity of one product also depends on the inventory of the other. Trying to obtain an optimal policy via DP for such a case requires not only to consider the stock of one product, but also that which serves as a substitute increasing the state space. This is where our research question comes in. Can High Performance Computing (HPC) aid to consider such cases which imply a larger state space?

The question of applying GPUs for value iteration (VI) in Markov Decision Problems (MDP) has been investigated by several researchers. Johannson [10] provides a proof of principle, where CUDA is used to map the VI process on a GPU platform. Chen and Lu [3] consider OpenCL for implementing a VI for a path finding problem. Herrera et al. [9] use HPC to implement a very large-scale MDP for the generation of traffic control tables. Ruiz and Hernández [14] use the developed concepts of [10] and claim a 90× speedup on an MDP model for crowd navigation. To the knowledge of the authors, no experience has been reported in literature on applying GPU implementations for VI to derive inventory control rules.

The contribution of this paper is to illustrate the MDP process for perishable inventory control first by a simplified retailer inventory control model of one product and

then to investigate how a two product case with a larger state space might be handled exploiting the use of GPUs. Therefore, Sect. 2 first describes a simplified one product inventory control problem and illustrates how the optimal solution can be derived by value iteration. Then, Sect. 3 discusses how we can handle a two product case with interaction of products caused by substitution. Section 4 describes the CUDA implementation of the algorithm. Section 5 depicts the evaluation results of the CUDA approach in comparison with a sequential version of VI. Finally, Sect. 6 provides the findings of our investigation.

2 Value iteration for a one product model

We first describe the decision situation and model for one product and then elaborate the optimal solution by value iteration. Section 2.3 provides an illustrative example.

2.1 Decision model for one product

In the first studied situation, the retailer receives the order placed the day before and inspects the inventory levels $(I_1, I_r, \ldots, I_M)$, where index r gives the remaining shelf life and M is the maximum shelf life when the product is received. The retailer orders a quantity Q, which will be received next morning for a cost of c per item. During the day, items are sold against a sales price $s > c$. Implicitly, like in the situation of the newsvendor boy, the retailer tries to minimize waste and lost sales. The daily profit over a long horizon T is given by

$$\Pi = \frac{1}{T}\sum_{t=1}^{T}(s\text{Sale}_t - cQ_t), \tag{2}$$

where the sales are determined by stochastic demand $\boldsymbol{d}_t$ from a Poisson distribution with mean μ:

$$\text{Sale}_t = \min\{\boldsymbol{d}_t, Y\}, \tag{3}$$

where $Y = \sum_{r=1}^{M} I_{rt}$ is the total inventory. The dynamics of the inventory depends on the so-called issuing of the products sold to the clients and we will assume that the retailer manages the shelves in such a way that the customers pick the oldest item first, i.e., First In First Out, FIFO. This means

$$I_{Mt} = Q_{t-1} \tag{4}$$

and

$$I_{rt} = \left(I_{r+1,t-1} - \left(\boldsymbol{d}_{t-1} - \sum_{m=1}^{r} I_{m,t-1}\right)^{+}\right)^{+}, \quad r = 1, \ldots, M-1. \tag{5}$$

Although not a direct performance indicator, we can measure the amount of waste as a percentage of the total order quantity that is generated by a certain policy:

$$Wasteperc = \frac{\sum_{t=1}^{T}(I_{1t} - d_t)^{+}}{\sum_{t=1}^{T} Q_t}. \tag{6}$$

A practical policy (see [6]) to decide on the order quantity Q_t may be a base stock policy, where the retailer also counts the amount of oldest items I_1 in order to have a rough estimate of the expected number of products perished at the end of the day, given as $\lceil (I_1 - \mu)^+ \rceil$. So this waste conscious base stock policy will provide an advice to order quantity

$$Q_t = \left(S - \sum_{r=1}^{M} I_{rt} + \lceil (I_{1t} - \mu)^+ \rceil \right)^+ . \tag{7}$$

Simulation of the system can be used to find the best value for S, which optimizes the daily profit (2).

2.2 Value iteration to reach the optimal policy

The question is whether there might be a better rule than (7) when the retailer measures the complete age distribution of its inventory $(I_1, I_r, \ldots, I_M)$. Actually, the described system behaves as a so-called Markov Decision Process (MDP), [13]. Given that there is a maximum amount to be ordered $\overline{Q}$, the state space is given by $I_1 = 0, \ldots, \overline{Q}$, $\mathcal{S} = \{0, \ldots, \overline{Q}\}^M$ consisting of $N = (\overline{Q}+1)^M$ elements. For each inventory situation $I \in \mathcal{S}$, we would like to know how many items $Q(I)$ to order.

Let $F(Q, I, d)$ be the function that tells us given inventory state I, order Q and demand d, what will be the inventory state next morning. Under certain circumstances, the optimal order amount $Q^*(I)$ can be characterized by the MDP theory, [1]. For the optimal policy, there exists a matrix $V(I)$ called the value function and a scalar π such that $\forall I \in \mathcal{S}$

$$V(I) + \pi = s\text{Esale}(I) + \min_{q=0}^{\overline{Q}} \left[\sum_{d=0}^{\infty} p_d V(F(q, I, d)) - cq \right], \tag{8}$$

where p_d is the probability that demand has value d and expected sales having $Y = \sum_{r=1}^{M} I_r$ total in stock is

$$\text{Esale}(I) = \sum_{d=1}^{Y} d \times p_d. \tag{9}$$

In this case, the valuation matrix V is not unique in the sense it is additive invariant, i.e., one can add constants to it, but the value π is unique and coincides with the optimal daily profit (2). An optimal policy is a solution of

$$Q^*(I) = \text{argmin}_q \left[\sum_{d=0}^{\infty} p_d V(F(q, I, d)) - cq \right]. \tag{10}$$

With respect to the infinite sum in (8) and (10), one should keep in mind that $F(q, I, d) = (0, 0, \ldots, q)$ as soon as $d \geq Y$, so only the cumulative distribution value of the total inventory is relevant for $d \geq Y$.

Algorithm 1 Pseudocode of VI for one product inventory control

1: Set vector elements V_j to Esale$_j$ for $0 = 1, \ldots, N-1$
2: **repeat**
3: Copy vector V into vector W
4: **for** $j = 0, \ldots, N-1$ **do**
5: Determine expected sales for state j, $Esale_j$
6: **for** $q = 0, \ldots, \overline{Q}$ **do**
7: **for** all demand realisations (events) d **do**
8: Retrieve W_k, with state $k = F(q_a, q_b, j, d)$
9: $V_j = \text{Esale}_j + \max_q[\sum_d p_d W_k - cq]$
10: **until** $\max_j(V_j - W_j) - \min_j(V_j - W_j) < \epsilon$

How to obtain a good valuation V and the corresponding optimal policy Q^*? This can be done by a fixed point idea about (8) with respect to value π called value iteration (VI). Although terminology and concepts are more extensive, from a computational point of view it is sufficient to think in those terms. It is also convenient to think of multi-dimensional matrix V as an N-dimensional vector where the states are ordered $j = 0, \ldots, N-1$. This gives the possibility to predefine the values Esale$_j$ for the expected sales when arriving at the inventory state corresponding to j.

One way to deal with VI (see Algorithm 1) is to copy a current valuation vector V into a vector W and determine a new valuation V according to

$$V_j = \text{Esale}_j + \min_q \left[\sum_d p_d W_k - cq \right], \quad j = 0, \ldots, N-1, \tag{11}$$

where k is the state index related to state $F(q, I, d)$. The value iteration should lead to convergence toward $\pi = V_j - W_j$, for all $j = 1, \ldots, N$. Convergence to the scalar π is measured by the so-called

$$\text{span}(V, W) = \max_j(V_j - W_j) - \min_j(V_j - W_j).$$

The iterative procedure of Algorithm 1 stops whenever span(V, W) is smaller than a pre-specified value ϵ, which indicates the accuracy in estimating π [8,9]. Figure 1 illustrates the computation for an inventory situation with maximum shelf life $M = 3$ from state $I = (3, 2, 1)$. Three possible decisions are sketched. If demand exceeds total inventory $Y = 6$ with probability $P(\boldsymbol{d} \geq 6)$ we have that we arrive at $F(q, I, d) = (0, 0, q)$.

2.3 Illustration for one product case

As an example we consider an instance where sales price is $s = 1$, unit cost is $c = .5$ and the average daily demand is $\mu = 5$ and the maximum shelf life is only $M = 2$. For this instance, the optimal order-up-to value of policy (7) is $S = 13$. This provides a daily profit of $\Pi = 2.195$ and the percentage of waste on the total order quantity is $Wasteperc = 7.33$ based on a simulation of 400,000 periods.

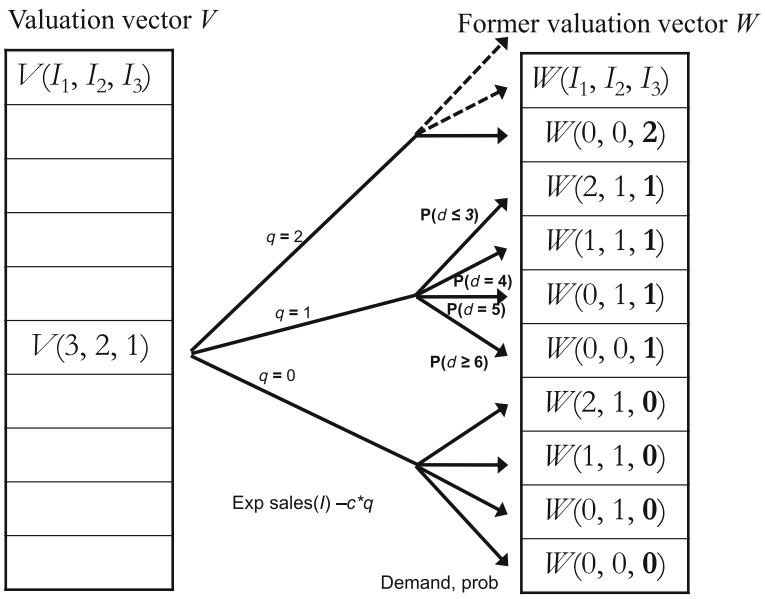

Fig. 1 Dependence of data in value iteration, one product and shelf life $M = 3$

To start the VI algorithm, we have to define the maximum order quantity $\overline{Q}$. This can be done by considering a newsvendor who orders for two periods following Equation (1), which provides $\overline{Q} = 9$. Using $\epsilon = 10^{-4}$, the algorithm converges in 12 iterations. Interestingly enough, the optimal policy Q^* never orders more than 7 units for this instance. The fixed value $\pi = 2.215$ corresponds to the simulation result of this policy, which is very close to the optimal base stock policy. However, the waste is reduced to $Wasteperc = 5.78$.

In order to challenge the algorithm, we increase the shelf life to $M = 3$, where the newsvendor would order 15 units corresponding to the optimal base stock $S = 15$ and leading to $N = 4096$ states. The algorithm converges in 15 iterations. Where the base stock policy gives a daily profit of $\Pi = 2.39$ and a $Wasteperc = 2.63$, the optimal policy is very near for this case with $\Pi = 2.40$ and a $Wasteperc = 2.53$.

The value iteration gets more challenge when we increase the shelf life to $M = 4$ where $\overline{Q} = 20$ for this case. Although this provides a computational challenge, the optimal value $\pi = 2.47$ corresponds practically to the average profit Π of the base stock policy with order-up-to level $S = 16$. So, for larger values of the shelf life, the systems starts to behave more like a non-perishable product and the order quantities get larger.

We also studied the smaller case of shelf life $M = 2$ with a LIFO withdrawal behavior, i.e., the clients first pick the freshest product. In that case, the optimal policy starts to behave periodic; order $Q = 10$ and next period order nothing, in order to avoid waste. For such cases, actually the Markovian behavior is different than used for the analysis in (8).

3 Handling two products due to substitution interaction

Recently, an interesting question has been raised where two products interact with each other due to substitution of one product by the other when one is out of stock. Several papers [2,4,5,16] try to deal with this phenomenon and show that for a variety of models, one gains by considering the two products simultaneously instead of individually. For our case, not only the state space becomes much larger, but we also have to find two order quantities simultaneously. We extend our analysis to a case or two products, a and b with the corresponding data $\mu_a, \mu_b, s_a, s_b, c_a, c_b$ and a probability γ for the binomial distribution Binom(n, γ) that models the amount of substitution demand for product a when product b is out of stock.

3.1 Value iteration for two products simultaneously

The decision becomes more cumbersome, not only because the state space becomes twice as big, but we also have to derive an order policy for Q_a and Q_b simultaneously; $[Q_a, Q_b](I_{a1}, I_{ar}, \ldots, I_{aM}, I_{b1}, I_{br}, \ldots, I_{bM})$. Moreover, the demand distribution for product a depends now not only on the stochastic demand $\boldsymbol{d}_a$, but also on the demand $\boldsymbol{d}_b$ as far as that exceeds the stock level $y = \sum_{r=1}^{M} I_{br}$. Let us call the substitution demand $\boldsymbol{u}$, which we consider for the case that $\boldsymbol{d}_b \geq 1$. So, on the one hand, no substitution takes place if $\boldsymbol{d}_b \leq y$ with $P(\boldsymbol{d}_b \leq y)$ and for those events $\boldsymbol{d}_b \geq y + 1$, the probability on substitution demand u is given by

$$pu(u, y) = p(\boldsymbol{u} = u|y) = \sum_{x=u}^{\infty} P(\boldsymbol{d}_b = x + y) \times Bin(u, x, \gamma), \quad u \geq 1 \tag{12}$$

where $Bin(u, x, \gamma)$ represents the probability mass function of the binomial distribution, i.e., the chance that there is u demand for substitution, when the inventory is exceeded by x out of stock for mean γ.

A difficulty in the analysis is to distinguish the events $\{\boldsymbol{d}_a = 0, \ldots, \infty, \boldsymbol{d}_b = 0, \ldots, y\}$ where the random variables can be considered independent as no substitution takes place and the events where there is inventory of product a and substitution may take place; $\{\boldsymbol{d}_a = 0, \ldots, \sum_r I_{ar} - 1, \boldsymbol{u} = 0, \ldots, \sum_r I_{ar} - \boldsymbol{d}_a\}$. For the latter case $(\sum_r I_{ar} \geq 1)$, one can derive the probability of having $d_b > y$ and a total demand of z for product a:

$$pz(z, y) = \sum_{x=0}^{z} P(\boldsymbol{d}_a = x) \times pu(z - x, y) \tag{13}$$

to be used to determine the state transition $F(Q_a, Q_b, I_a, I_b, d_a, d_b)$. Basically, in FIFO issuing, the dynamics is determined by (4) and (5) taking now the random events $\boldsymbol{d}_{at}, \boldsymbol{d}_{bt}$ and $\boldsymbol{u}_t$ into account as demand for the two products. To measure the profit, one should take care of a potential difference in sales price. The generation of waste according to (6) can be measured for both products individually. The algorithm for the two product case is sketched in Algorithm 2.

Algorithm 2 Pseudocode of VI for substitution of d_b by I_a

```
Set vector elements V_j to Esale_j for 0 = 1, ..., N − 1
repeat
    Copy vector V into vector W
    for j = 0, ..., N − 1 do
        Determine expected sales for state j, Esale_j
        for q_a = 0, ..., Q̄_a do
            for q_b = 0, ..., Q̄_b do
                for all Nre demand and substitution realisation d do
                    Retrieve W_k, with state k = F(q_a, q_b, j, d)
        V_j = Esale_j + max_{q_a,q_b}[Σ_d p_d W_k − c_a q_a − c_b q_b]
until max_j(V_j − W_j) − min_j(V_j − W_j) < ε
```

3.2 Two product instances

We elaborated first a symmetric case where both products have the same cost $c = .5$, sales price $s = 1$ and demand parameters values $\mu = 5$ for shelf life $M = 2$. This implies a 4-dimensional state space $(I_{a1}, I_{a2}, I_{b1}, I_{b2})$. The willingness to substitute product b by product a when being out of stock is $\gamma = .5$. As the maximum order quantity is set on $\overline{Q} = 10$, the state space contains 14,651 states and in each iteration, for each state 121 order quantity combinations are evaluated generating the corresponding transition probabilities. Simulating the system with (best) order-up-to levels $S_a = 13$ and $S_b = 12$ generates a daily profit of $\Pi = 4.479$ and the percentage of waste on the total order quantity is $Wasteperc_a = 6.26$ and $Wasteperc_b = 5.23$ based on a simulation of 400,000 periods. The value iteration process converges in 12 iterations to an accuracy of $\epsilon = 10^{-4}$. Simulation of the generated order quantities $[Q_a, Q_b](I_a, I_b)$ reveals a daily profit of $\Pi = 5.509$ and less waste generation; the percentage of waste on the total order quantity is $Wasteperc_a = 5.97$ and $Wasteperc_b = 4.14$. This illustrates that it is worth the trouble to look into the generation of better order policies by value iteration for highly perishable products when there is a willingness to substitute a product when it is out of stock.

3.3 Complexity of the algorithm

To analyze the computational burden of the value iteration algorithms, we simply measured the number of states and computational time of a Matlab implementation for several instances. A summary is provided by Table 1.

The computational burden (complexity) of Algorithm 2 for the case of two products with a shelf life of M is determined by the necessary number of iterations up to convergence, the number of states $(\overline{Q}_a + 1)^M \times (\overline{Q}_b + 1)^M$ and the number of relevant events in each state. We will denote the number of relevant events by $Nre = \sum_{r=1}^{M} I_{ar} + I_{br}$. We can observe that for shelf life $M = 3$ the Matlab code becomes intractable when the mean demand values are still realistic.

Table 1 Number of states N and computational time (in seconds or minutes) of a Matlab implementation for several instances with $s_a = s_b = 1$, $\gamma = .5$ and $c_a = c_b = .5$ on a Intel i5 with 8GB RAM

Instance								
Nr products	1	1	1	2	2	2	2	2
Shelf life M	2	3	4	2	2	3	3	3
$\overline{Q}_a$	10	15	20	10	14	6	6	15
$\overline{Q}_b$				10	10	3	6	15
μ_a	5	5	5	5	7	2	2	5
μ_b				5	5	1	2	5
N	121	4096	194,481	14,461	27,225	21,952	117,649	16.8×10^6
Comp time	.03 s	2.1 s	9 min	2 min	6 min	6 min	99 min	xxx

Largest case could not be solved

4 Parallelization of value iteration

In order to investigate a parallelization of VI, Algorithm 3 is constructed, which handles the states with respect to the number of relevant events *Nre* in line 8 of Algorithm 2.

Algorithm 3 depicts the details of the sequential code that we have considered to run the VI. For the case where no stock is available, indicated by $j = 0$, no stochastic event is relevant. For the states without inventory of product a, i.e., where $I_a = 0$, no substitution behavior is possible. Therefore, our intention has been to compute those states in parallel where substitution takes place ($\sum_r I_{ar} > 0$) because they require more than 95% of the total runtime. Running Algorithm 3 reveals that the most computationally demanding part are the lines 14–19 which compute the valuation $Fv(q_a, q_b)$ of an order decision (q_a, q_b). In order to tackle the parallelization, we have considered a GPU approach using Compute Unified Device Architecture (CUDA) interface.

CUDA is the parallel interface introduced by NVIDIA to develop parallel codes using C or C++ language. CUDA provides the SIMT (Single Instruction, Multiple Threads) programming model to exploit the GPU [12]. The programmer has to take several features of the architecture into account, such as the topology of the multiprocessors and the management of the memory hierarchy. For the execution of the program, the CPU (called host in CUDA) performs a number of kernel calls to the device. The input/output data to/from the GPU kernels are communicated between the CPU and the 'global' GPU memories by the PCI Express bus.

Algorithm 4 shows the host pseudocode of the GPU version that we have implemented. The algorithm implements lines 14–19 of Algorithm 3 to be computed on a GPU. Notice that the value of W is the same for all the states (j) of Algorithm 3, therefore, only one copy of W from CPU memory to GPU memory is required (before line 6 of Algorithm 3) in the GPU version.

In Algorithm 4, the number of threads and blocks are calculated and the call to GPU kernel (*GPUSubstitution*) is carried out. After that, a communication between the GPU and the CPU memory is necessary to transfer the computed value $Fv(q_a, q_b)$.

Algorithm 3 Details of the Pseudocode of sequential VI for two products

1: Set vector elements V_j to Esale$_j$ for $j = 1, \ldots, N$
2: **repeat**
3: Copy vector V into vector W
4: Determine states k corresponding to states $(0, \ldots, 0, q_a, 0, \ldots, 0, q_b)$
5: $V_0 = \min_{qa,qb} W_k$
6: **for** $j = 0, \ldots, N-1$ **do**
7: Determine expected sale $Esale_j$ for state j
8: Determine I_a, I_b corresponding to j
9: Set $Nre = (\sum_r I_{ar} + 1) \times (\sum_r I_{br} + 1)$
10: **if** $\sum_r I_{ar} = 0$ **then** ▷ **No substitution can take place**
11: Determine states $k = F(q_a, q_b, (0, \ldots, 0, I_{b1}, I_{b2}, \ldots, I_{bM}), d_b)$
12: $Fv(q_a, q_b) =$ Esale$_j + \sum_{d_b} p_d W_k - c_a q_a - c_b q_b$
13: **else** ▷ **we might have substitution**
14: **for** $q_a = 0, \ldots, \overline{Q}_a$ **do**
15: **for** $q_b = 0, \ldots, \overline{Q}_b$ **do**
16: **for** all Nre demand and substitution realizations d, u **do**
17: Determine states $k = F(q_a, q_b, j, d_a, d_b, u)$
18: $Fv(q_a, q_b) =$ Esale$_j + \sum_d p_d W_k - c_a q_a - c_b q_b$
19: $V_j = \max_{q_a,q_b}[Fv(q_a, q_b)]$
20: **until** $\max_j(V_j - W_j) - \min_j(V_j - W_j) < \epsilon$

Algorithm 4 GPU host pseudocode computing future values Fv for substitution

1: $nthreads \leftarrow 32\lceil \frac{Nre}{32} \rceil$
2: $nblocks \leftarrow (\overline{Q}_a + 1) \times (\overline{Q}_b + 1)$
3: $Fv(q_a, q_b) \leftarrow GPUSubstitution <<< nblocks, nthreads >>>$ ▷ Alg. 5
4: Communicate Fv from GPU memory to CPU memory
5: $V_j =$ Esale$_j + \max_{q_a,q_b}(Fv(q_a, q_b) - c_a qa - c_b q_b)$

Algorithm 5 presents the *GPUSubstitution* kernel which computes Fv in parallel. We spawn as many blocks as combinations (q_a, q_b), i.e., up to a total of $(\overline{Q}_a + 1) \times (\overline{Q}_b + 1)$ blocks, and each block contains, at least, as many threads as combinations (d_a, d_b), i.e., a total of $32\lceil \frac{Nre}{32} \rceil$ threads. Each thread computes the contribution to Fv of a combination (d_a, d_b) for block (q_a, q_b) and stores the result in a per-block shared memory array called $thvfuture$. Finally, the first thread of each block adds all these partial values and writes the final result for (q_a, q_b) in a global array called Fv.

5 Evaluation results

The value iteration for the two-product instance described in Sect. 3 has been implemented using C and CUDA. The used platform is a Bullx R421-E4: 2 Intel Xeon E5 2620v2 processor with 6 cores each and 64 GB of RAM. It is connected to 2 NVIDIA K80. Each NVIDIA K80 has two Kepler GK210 GPUs. The characteristics of each NVIDIA K80 are given in Table 2. The compilation is done with the '-O3' optimization flag. The platform runs Ubuntu 16.04 LTS and CUDA Toolkit 8.

For the investigated instance with $\mu_a = \mu_b = 5$, we first measured the number of relevant events *Nre* (see line 5 of Algorithm 5) for each state in one repeat-until

Algorithm 5 *GPUSubstitution* kernel computes Fv

Require:
$W, p_d, Esale_j, c_a, c_b, I_a, I_b, \overline{Q}_a, \overline{Q}_b$

1: **__shared__** $thvfuture$ ▷ Block-scope shared array for Fv reduction
2: $q_a \leftarrow blockIdx.x/(\overline{Q}_a + 1)$
3: $q_b \leftarrow blockIdx.x\%(\overline{Q}_a + 1)$ ▷ q_a and q_b calculated from block ID
4: $dadb \leftarrow threadIdx.x$ ▷ Block-scope thread ID
5: $Nre \leftarrow (\sum_r I_{ar} + 1) \times (\sum_r I_{br} + 1)$
6: **if** $dadb < Nre$ **then** ▷ If this thread has a valid combination of d_a and d_b
7: $d_a = dadb\%(\sum_r I_{ar} + 1))$ ▷ d_a and d_b calculated from local thread ID
8: $d_b = dadb/(\sum_r I_{ar} + 1))$
9: Determine states k corresponding to $k = F(q_a, q_b, j, d_a, d_b)$
10: $thvfuture[dadb] = \sum_d p_d W_k$ ▷ Every thread updates its $thvfuture$
11: **__syncthreads** ▷ Synchronization point for all threads in each block
12: **if** threadIdx.x == 0 **then** ▷ For a single thread of every block
13: $ivfuture = q_a(\overline{Q}_b + 1) + q_b$
14: $svfuture = Esale_j - c_a q_a - c_b q_b$
15: **for** $i = 1, \ldots, Nre$ **do** $svfuture+ = thvfuture[i]$
$Fv[ivfuture] = svfuture$ ▷ Update Fv
16: **return** $Fv(q_a, q_b)$

Table 2 Characteristics of the NVIDIA K80

	NVIDIA K80
Peak performance (double prec.) (TFlops)	2.91
Peak performance (simple prec.) (TFlops)	8.74
Device memory (GB)	24
Boost clock rate (MHz)	875
Memory bandwidth (GB/s)	480
Multiprocessors	13
CUDA cores	4992
Compute capability	3.7

iteration. The result, Fig. 2, illustrates the strong unbalance among the different state values $j = 0, \ldots, N-1$ due to the difference of the values of I_{a1}, I_{a2}, I_{b1} and I_{b2}. In order to balance the workload of the substitution stage, the number of blocks and threads is dynamically defined at every iteration (lines 1 and 2 of Algorithm 4).

In order to measure the runtime of the sequential and the GPU code of VI, we have generated 4 instances of the problem where μ_a and μ_b are varied and the willingness to substitute $\gamma = .5$, sales price $s_a = s_b = 1$ and cost $c_a = c_b = .5$ are kept constant. The instances are given in Table 3.

Table 4 shows the results of the experiments where the described parallel version is compared to the sequential C code for the four instances in Table 3. For each problem, we run the code for 100 iterations in order to have an accurate execution time and all converged before this maximum has been reached. As can be observed from Table 4, the parallel version improves the sequential version in a factor up to 11.7×. An interesting aspect is that due to the use of GPU computing, the Substitution part has considerably reduced its runtime. Specifically for $P4$ (the largest problem), *GPUSubstitution* is

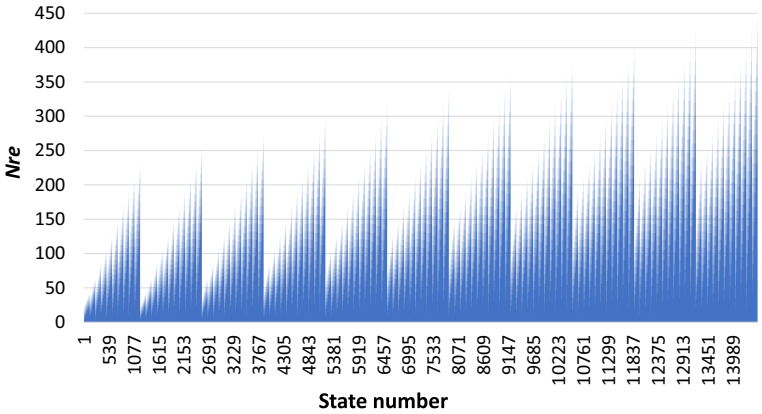

Fig. 2 Number of relevant events *Nre* (see line 5 of Algorithm 5) for each state in one VI iteration to show the unbalance among the state values $j = 0, \ldots, N-1$

Table 3 Instances $P1$ to $P4$ with varying mean demand μ_a, μ_b

	$P1$	$P2$	$P3$	$P4$
μ_a	5	5	6	7
μ_b	5	6	6	7
$\overline{Q}_a$	11	11	13	14
$\overline{Q}_b$	11	13	13	14
N	14,461	20,449	28,561	38,416

The sales price $s_a = s_b = 1$ and cost $c_a = c_b = .5$ are kept constant. Willingness of substitution is $\gamma = .5$. Instead of running the repeat until an accuracy, to measure the differences, we run the algorithm for 100 iterations

Table 4 Total runtime in seconds of the sequential and the GPU code to compute the VI algorithm

	$P1$	$P2$	$P3$	$P4$
Seq. code				
Total runtime of Algorithm 3	357.89	698.20	1358.04	4241.66
Substitution (lines 14–19)	351.31	688.71	1344.82	4217.15
GPU code				
Total runtime	91.29	136.84	214.71	361.49
Communication host to device	0.00	0.00	0.01	0.01
Communication device to host	21.55	23.35	43.63	56.69
GPUSubstitution	39.94	67.29	111.99	215.54
Total Communic. GPU/CPU	21.56	23.35	43.64	56.70
% Communications	23.62	17.06	20.32	15.69
% *GPUSubstitution*	43.75	49.18	52.16	59.62
% Other parts of VI	32.63	33.76	27.52	24.69

Row Substitution (*GPUSubstitution*) is runtime required by lines 14–19 of Algorithm 3 (line 3 of Algorithm 4). Communication: required runtime for communication between CPU/GPU. %: percentage of the runtime of the CUDA kernel, the communication and other functions

19.6× faster than the sequential substitution. What can also be observed is that when problems become larger, a larger part of the computation can be done by the GPU.

6 Conclusions

Value iteration is an algorithm that can be used to derive optimal inventory control order policies. The computational challenge becomes interesting when a number of products and the age distribution is considered simultaneously. The latter aspect is of interest in the case of perishable products with a low mean demand that sometimes may be substituted by other products when being out of stock due to a revealed customer willingness to do so. This paper describes a simple model and illustrates the underlying algorithm.

The complexity increases with the shelf life of a perishable product, but on the other hand leads to less interesting policies, as simple order-up-to level policies give practically the same performance in profit and waste generation. The challenging case is where several highly perishable products may be substituted by another, because ordering product a also requires looking into inventory levels (and age distribution) of product b, which can be substituted by a. Basically, dimension grows more than quadratically with the shelf life of the products.

Therefore, we developed a parallel version of the corresponding value iteration algorithm based on computing on a GPU architecture the part of the model which has the highest computational cost. This CUDA code tries to efficiently compute the nested-loops required for calculating the substitution of one product by another. Numerical results based on a case of only two products show that the parallel algorithm may improve the performance of the sequential value iteration in a factor that reaches 11.7×. Our future work will be focused on the parallel implementation of the relevant two product inventory management where shelf life is three.

References

1. Bellman R (1957) A Markovian decision process. J Math Mech 6(5):679–684
2. Buisman M, Haijema R, Hendrix EMT (2018) On the delta service level for demand substitution in inventory control. IFAC-PapersOnLine 51:1660–1665
3. Chen P, Lu L (2013) Markov decision process parallel value iteration algorithm on GPU. In: Proceedings of 2013 International Conference on Information Science and Computer Applications. Atlantis Press, pp 299–304
4. Chen X, Feng Y, Keblis MF, Xu J (2015) Optimal inventory policy for two substitutable products with customer service objectives. Eur J Oper Res 246(1):76–85
5. Gürler U, Yilmaz A (2010) Inventory and coordination issues with two substitutable products. Appl Math Model 34(3):539–551

6. Haijema R, Minner S (2016) Stock-level dependent ordering of perishables: a comparison of hybrid base-stock and constant order policies. Int J Prod Econ 181(PA):215–225
7. Hendrix EMT, Haijema R, Rossi R, Pauls-Worm KGJ (2012) On solving a stochastic programming model for perishable inventory control. In: Murgante B et al (eds) Computational science and its applications-ICCSA 2012. Springer, Heidelberg, pp 45–56
8. Hendrix EMT, Ortega G, Haijema R, Buisman M, García I (2018) On computing optimal policies in perishable inventory control using value iteration. In: Proceedings of the 18th CMMSE, pp 1–6
9. Herrera JFR, Hendrix EMT, Casado LG, Haijema R (2014) Data parallelism in traffic control tables with arrival information. In: Euro-Par 2014: Parallel Processing Workshops. Springer, Cham, pp 60–70
10. Johannson A (2009) GPU-based Markov decision process solver. Master's thesis, Reykjavik University, Iceland
11. Minner S (2000) Strategic safety stocks in supply chains. Springer, Berlin
12. NVIDIA Corporation: CUDA C PROGRAMMING GUIDE PG-02829-001_v9.2 (2018)
13. Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming, 1st edn. Wiley, New York
14. Ruiz S, Hernández B (2015) A parallel solver for Markov decision process in crowd simulations. In: 2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI), pp 107–116
15. Silver E, Pyke D, Peterson R (1998) Inventory management and production planning and scheduling. Wiley, New York
16. Yadavalli V, Sundar D, Udayabaskaran S (2015) Two substitutable perishable product disaster inventory systems. Ann Oper Res 233(1):517–534