



**HAL**  
open science

# Accelerating viability kernel computation with CUDA architecture: application to bycatch fishery management

A. Brias, Jean-Denis Mathias, G. Deffuant

## ► To cite this version:

A. Brias, Jean-Denis Mathias, G. Deffuant. Accelerating viability kernel computation with CUDA architecture: application to bycatch fishery management. *Computational Management Science*, 2016, 13 (3), pp.371-391. 10.1007/s10287-015-0246-x . hal-01852648

**HAL Id: hal-01852648**

**<https://hal.science/hal-01852648>**

Submitted on 2 Aug 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

<b>Noname manuscript No.</b> (will be inserted by the editor)
--

---

1 **Accelerating viability kernel computation with**  
2 **CUDA architecture : application to bycatch fishery**  
3 **management**

4 **Antoine Brias · Jean-Denis Mathias ·**  
5 **Guillaume Deffuant**

6 Received: date / Accepted: date

7 **Abstract** Computing a viability kernel consumes time and memory resources  
8 which increase exponentially with the dimension of the problem. This curse  
9 of dimensionality strongly limits the applicability of this approach, otherwise  
10 promising. We report here an attempt to tackle this problem with Graphics  
11 Processing Units (GPU). We design and implement a version of the viabil-  
12 ity kernel algorithm suitable for General Purpose GPU (GPGPU) computing  
13 using Nvidia's architecture, CUDA (Computing Unified Device Architecture).  
14 Different parts of the algorithm are parallelized on the GPU device and we  
15 test the algorithm on a dynamical system of theoretical population growth.  
16 We study computing time gains as a function of the number of dimensions and  
17 the accuracy of the grid covering the state space. The speed factor reaches up  
18 to 20 with the GPU version compared to the Central Processing Unit (CPU)  
19 version, making the approach more applicable to problems in 4 to 7 dimen-

---

A. Brias  
Iristea, UR LISC Laboratoire d'ingénierie des systèmes complexes, 24 avenue des Landais-  
BP 20085 Aubière, F-63172 France  
Tel.: +123-45-678910  
Fax: +123-45-678910  
E-mail: antoine.brias@irstea.fr

20 sions. We use the GPU version of the algorithm to compute viability kernel of  
21 bycatch fishery management problems up to 6 dimensions.

22 **Keywords** Viability kernel · Dynamic programming · CUDA · GPU · Fishery  
23 management

24 **Mathematics Subject Classification (2000)** 90B50 - Management Deci-  
25 sion Making, including Multiple Objectives · 90C39 - Dynamic Programming

## 26 1 Introduction

27 Viability theory provides mathematical and numerical tools and concepts for  
28 maintaining a dynamical system within a set of states (called the constraint  
29 set). This theory has numerous potential applications in food processing, fi-  
30 nance, economics, environment (Sicard et al, 2012; Béné et al, 2001; Doyen  
31 et al, 2012; Bernard and Martin, 2013; Andrés-Domenech et al, 2014; Chapel  
32 et al, 2008, 2010; Mathias et al, 2015). A essential step in this approach lies in  
33 the computation of the viability kernel. This is the set of states for which there  
34 exists a control policy that keeps the system in the constraint set for some (fi-  
35 nite or infinite) time. Since the 90's, several algorithms have been developed  
36 to compute viability kernels in different application fields (Saint-Pierre, 1990;  
37 Bokanowski et al, 2006). Some of them are adapted to finite horizon prob-  
38 lems (Djeridane and Lygeros, 2008), linear dynamics (Kaynama and Oishi,  
39 2013) or particular kinds of problems such as single output nonlinear control  
40 systems affine in the control (Turrieff and Broucke, 2009; Mattioli and Artiou-  
41 chine, 2003). The concepts have been extended to the stochastic case (Doyen  
42 and De Lara, 2010; Rougé et al, 2013), in which dynamics includes random  
43 variables accounting for uncertainties. However, most of them are limited by  
44 their computational complexity. When the number of dimensions increases,

45 the required computation time and memory increase exponentially because  
46 they are mainly based on a grid covering the state space.

47 This curse of dimensionality is a moot point in a number of domains,  
48 like data mining, numerical analysis (Donoho, 2000) and multiple approaches  
49 aim at mitigating it and at speeding up viability kernel computing. For in-  
50 stance, Bonneuil (2006) attempts to handle large dimensional state space with  
51 computing control policies on a given time horizon with simulated annealing.  
52 Deffuant et al (2007) use Support Vector Machines for approximating viability  
53 kernels. Maidens et al (2009) try to overcome this curse by using Lagrangian  
54 methods, which do not call for a grid. It offers particularly interesting per-  
55 formances for linear dynamics. Designing new algorithms is a crucial way to  
56 reduce the curse of dimensionality, but existing technological solutions can  
57 also lead to important gains of time. Some programs like Vikaasa, used in  
58 by-cash fishery management, embed a multi-core implementation of viability  
59 kernel computation (Krawczyk et al, 2013). However, in recent years, another  
60 technical improvement arose which seems well adapted for the viability kernel  
61 problematic : the GPU parallel computing technology. Historically, GPUs are  
62 electronic components used in computer graphics hardware. They are getting  
63 more widely used to accelerate computations in many fields (Mametjanov et al,  
64 2012; Cekmez et al, 2013; Sabo et al, 2014). Due to its architecture, the GPUs  
65 solution is advantageous for problems for which a parallel execution is possi-  
66 ble (Goldsworthy, 2014). It often exceeds the capacity of computing clusters  
67 in terms of performance and for a lower cost. Thus, NVIDIA has created the  
68 CUDA platform to offer tools and to allow programmers to use the parallel  
69 architecture of graphic cards. The basic point is to separate the program in  
70 two parts : the serial code executed by the host (the Central Processing Unit  
71 (CPU)), and the parallel code, executed in the GPU threads across multiple  
72 processing elements.

73 In this paper, we use the standard viability algorithm (Saint-Pierre, 1990)  
 74 which can be considered as a particular case of dynamic programming. To  
 75 illustrate our work, the implemented algorithm is tested on a basic multidimensional model of population dynamics. In particular, we compare the computation time between the parallelized version and the sequential one. Finally,  
 76 the algorithm is used to compute the viability kernel in a case of bycatch  
 77 fishery management.

## 80 2 Viability theory and dynamic programming

### 81 2.1 Viability kernel

82 We consider a discrete dynamics in which  $h$  is the function mapping the state  
 83 and the control at time  $t$  with the state at the next time step  $t + 1$ :

$$x(t + 1) = h(x(t), u(t)) \quad (1)$$

84 with  $x(t) \in \mathcal{X}$  the set of states,  $u(t) \in \mathcal{U}(x(t))$  which is a finite set of possible  
 85 controls allowing the regulation of the dynamics and  $h(.,.) : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{X}$ ,  
 86 associating a state  $x(t) \in \mathcal{X}$  and a control  $u(t) \in \mathcal{U}(x(t))$  with a successor  
 87  $x(t + 1) \in \mathcal{X}$ . For example, it associates the closest grid point of the successor  
 88 obtained with an Eulerian scheme defined with an appropriate local parameter.  
 89 We want to assess the viability kernel  $Viab_T(K)$ ; the set of initial states for  
 90 which a control strategy exists to maintain the system inside  $K$ , the set of  
 91 desirable states (the viability set), until  $T$  (a finite horizon). This is rewritten  
 92 in Eq.2 :

$$Viab_T(K) = \{x_0 \in K | \exists (u(0), u(1), \dots, u(T-1)), \forall t \in \{0, 1, \dots, T\}, x(t) \in K\} \quad (2)$$

93 with  $x(t)$  the state of the system at date  $t$  and  $u(t)$  the chosen control  
 94 value at  $t$ , which maintains the system in the viability set  $K$ .

95 Different algorithms are available for computing this set. Because of the  
 96 approximation of the dynamical system on a grid, these algorithms are sub-  
 97 mitted to the dimensionality curse: the necessary time and memory increase  
 98 exponentially with the number of dimensions of the problem. We choose to  
 99 focus on a dynamic programming algorithm which is very standard and well  
 100 suited for a parallel approach.

## 101 2.2 Dynamic programming

102 We assume that the dynamics of the system fulfils the condition ensuring the  
 103 convergence of the approximation to the actual kernel, when the resolution of  
 104 the discretization tends to 0 (Saint-Pierre, 1990). The dynamic programming  
 105 algorithm is a backward recursive algorithm and determines a value function  
 106 :  $V(t, x)$  at date  $t$ . The seed value at  $T$  is :

$$V(T, x) = \begin{cases} 1 & \text{if } x \in K \\ 0 & \text{if } x \notin K \end{cases} \quad (3)$$

107 In this case, the value function directly determines the viability kernel. At  $T$ ,  
 108 being viable is the property of belonging to the viability kernel. The linear  
 109 recurrence relation is given by :

$$\forall t \in \{0, T-1\}, V(t, x) = \begin{cases} \max_{u \in \mathcal{U}(x)} V(t+1, h(x, u)) & \text{if } x \in K \\ 0 & \text{if } x \notin K \end{cases} \quad (4)$$

110 Thus, at  $t = 0$ :

$$V(0, x) = \begin{cases} 1 & \text{if } x \in Viab_T(K) \\ 0 & \text{if } x \notin Viab_T(K) \end{cases} \quad (5)$$

111 The principles of dynamic programming are respected because, at each  
112 step time :

- 113 - each point of the grid is a subproblem ( $V$  has to be computed for each point);
- 114 - a value function has to be optimized independently for these points;

115 The value function  $V$  takes only two values : 0 and 1. We are looking  
116 for initial states, for which the value function at time horizon  $T$  is 1. Eq.4  
117 is equivalent to say that the value function  $V(t, x)$  is equal to 1 if a control  
118  $u \in \mathcal{U}(x)$  exists, maintaining the system in the constraint set  $K$ . Otherwise,  
119 the value function  $V(t, x)$  equals to 0. The backward dynamic programming  
120 equation defines the value function  $V(x, t)$ . By writing a max instead of a sup,  
121 we implicitly assume the existence of an optimal solution for each time  $t$  and  
122 state  $x$ . The algorithm involves two main steps:

- 123 – For each point of the grid, we store all the successors (the point of the grid  
124 at the next time step) obtained for each choice of control (the possible set  
125 of actions at each point is supposed finite).
- 126 – The second part performs iterations of kernel approximations by progres-  
127 sively excluding points of the grid until reaching a fixed point or a prede-  
128 fined number of time steps.

129 Thus we have the algorithm Alg.1 as follows :

**Algorithm 1** Detailed Dynamic Programming Viability Kernel Algorithm

---


$$\begin{array}{l}
\text{Storing successors : } \left\{ \begin{array}{l} i = 0 \\ K^0 = K \\ \text{store coordinates, } \forall x \in K \\ \text{store } h(x, u), \forall x \in K, \forall \mathbf{u} \in \mathcal{U}(x) \end{array} \right. \\
\text{Iterating kernel approximation : } \left\{ \begin{array}{l} \text{repeat} \\ K^{i+1} = \{x \in K^i \mid V(i+1, x) = 1\} \\ i = i + 1 \\ \text{until } K^{i+1} = K^i \text{ OR } K^{i+1} = \emptyset \text{ OR } i = T \end{array} \right. \\
Viab_T(K) = K^i
\end{array}$$


---

130 **2.3 Parallelizing dynamic programming algorithm**

131 Dynamic programming is well suited to parallelization in the two main parts.  
132 During the storing of successors, all successors  $x(t+1)$  of each state and  
133 each value of  $u$ , defined in Eq.1, can be computed separately. In the second  
134 part of the algorithm, it is possible to parallelize the update of the value  
135 function  $V(t, x)$  applied to each state (Eq.4). In both cases, computations are  
136 completely independent for one step of time. The parallelizing of all these  
137 calculations thus improves the performance in terms of computation time.  
138 Since GPUs have hundreds of processor cores running in parallel, they can be  
139 more efficient than using traditional CPU architectures.

140 **3 Using GPU for computing viability kernels**141 **3.1 GPU architecture**

142 As they are widely used in many scientific computing fields, graphics hardware  
143 programming for general purpose computing is well documented, this includes  
144 ways to write and debug the code (Langdon, 2011). Providing highly effective  
145 data parallelism, GPU cores contain multiple threads running at the same time



146 which can accelerate viability algorithms. The CUDA programming framework  
 147 enables the use of all the cores belonging to the GPU for general purpose com-  
 148 puting. Programs written in this paper use the C++ CUDA framework. A  
 149 GPU computing view consists in splitting the input data between the differ-  
 150 ent threads, and performing operations on these pieces of data, independently  
 151 and thus achieving parallelization. This kind of computing encourages to re-  
 152 think programs to make this parallelization possible. Fig.1 illustrates the ar-  
 153 chitecture of a GPU device which is seen as a grid housing blocks. Each block  
 154 involves a number of threads performing the computations. The GPU device  
 155 owns a global memory, each block has a memory shared by all of its threads,  
 156 and each thread owns a smaller local memory. Here, we use the global memory  
 157 to store big arrays of data with which all the threads can interact. In a block,  
 158 a thread is indexed over 3 dimensions (a block can be shown as a cube of  
 159 threads), thus the indexation has to be carefully done. Each thread and each  
 160 block are indexed at their level. In our implementation, we want to get the ID  
 161 *threadID* of each one-dimensional thread in a grid of 3D blocks. According to  
 162 the CUDA framework writing convention, this ID is given by :

$$\begin{aligned}
 threadID = & threadIdx.i + blockDim.i * blockIdx.i + \\
 & gridDim.i * blockIdx.j * blockDim.i + \\
 & gridDim.i * gridDim.j * blockIdx.k * blockDim.i
 \end{aligned} \tag{6}$$

163 Here, *threadIdx.i* means the coordinates *i* of *threadIdx*. *gridDim* contains  
 164 the dimensions of the grid (the grid owns  $gridDim.i \times gridDim.j \times gridDim.k$   
 165 blocks), *blockIdx* contains the block indexes within the grid (in 3 dimensions  
 166 : *blockIdx.i*, *blockIdx.j* and *blockIdx.k*), *blockDim* contains the dimensions  
 167 of the block (a block contains  $blockDim.i \times blockDim.j \times blockDim.k$  threads

168 in 3 dimensions) and *threadIdx* contains the thread indexes within the block  
169 (in 3 dimensions too, but for simplicity, only one is used here : *threadIdx.i*).  
170 These variables are mainly used in assigning the work per block and thread.  
171 The maximum values in each dimension in both *blockDim* and *gridDim* are  
172 GPU-dependent. At least, the *i* component needs to be declared. By default,  
173 the value for the *j* and *k* components is 1.

### 174 3.2 Overall strategy

175 The organization of the algorithm is summed up in Fig. 2. The figure shows the  
176 storing of trajectories and the iterating kernel approximation steps of Alg.1.  
177 The steps are :

- 178 - the initialization and setting of the GPU device (a);
- 179 - the initialization of the model parameters (b). The user inputs these param-  
180 eters;
- 181 - the initialization of CUDA parameters (number of blocks, number of threads  
182 by blocks in each dimension)(c);
- 183 - the creation of empty arrays, which will contain controlled evolutions and  
184 the value function of each point using the CUDA instruction *cudaMalloc*  
185 (d);
- 186 - the copy of parameters from host to the GPU device (e) using the CUDA  
187 instruction *cudaMemcpy* with the flag *cudaMemcpyHostToDevice*;
- 188 - the computation of successors of each point i.e. the storing of next point of  
189 the trajectory for each possible value of the control Alg.1. This operation is  
190 parallelized on the GPU and it is done in the first kernel function. Section  
191 3.3 detailed this part.
- 192 - the iterating kernel approximation step of Alg.1. The value function of each  
193 point is calculated and ran until the stop condition is reached. It is paral-

- 194     lelized in the second kernel function on the GPU. Section 3.4 detailed this  
 195     part.
- 196 - the copy of results from the GPU memory to the CPU (f) using the CUDA  
 197     instruction *cudaMemcpy* with the flag *cudaMemcpyDeviceToHost*;
  - 198 - The clear of the memory of the GPU and the finalization of the GPU device  
 199     (g), using the CUDA instruction *cudaFree*;
  - 200 - the deallocation of the CPU memory (h).

### 201 3.3 Parallelizing the computing and the storing of successors

202 The input of the algorithm is the number of points by dimension and each  
 203 dimension bound. The algorithm computes the coordinates of all the points  
 204 and their index. It can be parallelized since each point is independent from  
 205 the other. Thus, one thread on the GPU will compute the coordinates of one  
 206 point.

207 Now, we have to compute the all the successors of each point of the grid,  
 208 when applying the different values of the control (if the control can take con-  
 209 tinuous values then it is discretized). We can thus suppose that the control  
 210 takes *nbControls* values point of the grid. Then, on each thread, the coordi-  
 211 nates of one state  $X_i$  are sent, and all the *nbControls* possible evolutions from  
 212 this state are returned. Again, the number of threads is equal to the number  
 213 of points of the grid. This treatment is summed up on Fig. 3. There are four  
 214 steps in the GPU computation of the trajectories :

- 215 - the inputs are the values of the discretized control,  $U = (u_1, \dots, u_{nbControls})$   
 216     and the coordinates of all the points  $X = (X_1, \dots, X_{nbPoints})$ . These coordi-  
 217     nates were determined beforehand and remain in the GPU global memory;
- 218 - this data is sent on threads, each thread holding the computation of all the  
 219     controlled evolutions of only one point;

220 - each thread then sends back the result in the array  $H$ . This array contains the  
 221 index of the controlled evolutions. Moreover, it contains -1 if the evolution  
 222 doesn't belong to  $K$ . Thus, we store the index of  $i$ -th controlled evolution  
 223 of  $X$  in  $H(X, i)$ . As the number of threads is the same for each block, some  
 224 threads can be superfluous on the last block;

225 - once we get the controlled evolutions, we just have to copy them from the  
 226 GPU to the CPU and we apply the second part of the dynamic program-  
 227 ming algorithm.

### 228 3.4 Parallelizing the iterating kernel approximation

229 The second part of the algorithm computes the value function  $V$  for each point  
 230 until the stopping condition is reached. As shown on Fig. 4, we parallelize this  
 231 task. The global memory contains the value function array :  $v$  in which the  
 232 value function  $V$  of each point is stored; the successors computed at the first  
 233 step :  $H$ ; and a boolean value :  $CheckChange$  which states if there is no  
 234 change during the step. In this latter case, the algorithm stops. The current  
 235 value function  $V$  is stored in a temporary array  $vTmp$ . The treatment below  
 236 (Alg. 2) is applied to each thread (one point  $X$  by thread) :

---

**Algorithm 2** Update of the value function of  $X$  in the iterating kernel approximation.

---

If  $v(X)=1$  (We check if the point is viable, i.e., if the value function applied to it is equal to 1)

If  $\forall i \in [1, nbControls - 1]$ ,  
 $(H(X, i) = -1)$  OR  $(H(X, i) \geq 0$  AND  $v(H(X, i)) = 0)$   
 $vTmp(X) = 0$   
 $CheckChange = CheckChange + 1$

End If

End If

---

237 If the point is viable and if a viable successor exists, nothing is done, the  
238 point is still viable. But if none of its successors is viable, i.e. if none of its  
239 successors belongs to  $K$  (i.e.,  $H(X, i) = -1$ ) or if the value function  $V$  of the  
240 successors is equal to 0, then the point is considered non-viable : 0 is put in  
241 the appropriate cell of the array  $vTmp$ . This treatment represents the value  
242 function updating in dynamic programming. In this case, a change is made,  
243 then *CheckChange* is incremented. At the end of the step, the value function  
244 is transferred of  $vTmp$  to  $v$  and we check *CheckChange*. If it is equal to 0,  
245 the algorithm can stop. It happens after a sufficient number of iterations and  
246 we then get the approximation of the viability kernel.

## 247 4 Results

248 First, we test the GPU programming on a simple example of population dy-  
249 namics, in order to compare the performance with the sequential algorithm.  
250 Then, we use our parallel algorithm to treat a problem of bycatch fishery  
251 management up to 6 dimensions.

### 252 4.1 A multi-dimensional population model

253 One toy model is used here to evaluate the benefits of GPU programming. We  
254 consider the population model studied by Malthus and Verhulst. Malthusian  
255 ideas are the basis of the classical theory of population and growth (Ehrlich and  
256 Lui, 1997). This model is a simple dynamical system of population growth on a  
257 bounded space without any predator. The system state includes two variables :  
258 the size of the population and its evolution rate. Aubin and Saint-Pierre (2007)  
259 introduced a control on this evolution rate and adapted it to the management  
260 of renewable resources.

261 *4.1.1 The model*

262 As we want to analyze the performances of the parallelized algorithm on multi-  
 263 dimensional problems, we artificially increase dimensions of the system in  
 264 adding some independent variables in order to study the behaviour of the  
 265 program in different dimensions. The new model is as follows : we considered  
 266  $n$  states  $x_i$  representing the size of  $n$  populations, which all have the same evo-  
 267 lution rate  $y(t) \in [d, e]$ . The size of the populations must remain in  $K = [a, b]$ .  
 268 The control is applied on the derivative of the evolution rate at each time step  
 269 with the inertia bound  $c$ . The equations ruling the system in discrete time are  
 270 :

$$x_i(t+1) = x_i(t) + x_i(t)y(t), \quad i \in \{1, \dots, n\} \quad (7)$$

$$y(t+1) = y(t) + u(t) \quad (8)$$

271 with  $-c \leq u(t) \leq +c$ . The set  $K = [a, b]^n \times [d, e]$  is the viability constraint  
 272 set. For the rest of this work, bounds for  $K$  are set as  $a = 0.2$ ,  $b = 3$ ,  $c = 0.5$ ,  
 273  $d = -2$ ,  $e = 2$ . By default, the control is discretized in 5 values. Since the  
 274 parallelization simply breaks down processes that are already independent in  
 275 the usual algorithm, the discretization does not impact the accuracy of the  
 276 results.

277 The program is implemented in C++, using CUDA 5. The computer CPU  
 278 is an Intel® Xeon(R) CPU E5-2687W 0 @ 3.10 GHz x 16 and the GPU card  
 279 is a Tesla K20C with a computing capability of 3.5 and 2496 CUDA cores.

---

#### 4.1.2 Speed-up

Here we show a comparison between the two versions of the program applied to this model. The speed-up  $S$  is a measure evaluating the performance of the parallelization:

$$S = \frac{T_{seq}}{T_{par}} \quad (9)$$

where  $T_{seq}$  is the sequential algorithm execution time and  $T_{par}$  the execution time of the parallel one. The first one uses the usual algorithm; the second one uses the adapted algorithm to GPU computing.

We figure the evolution of the speed-up, in function of the size of the grid. In Fig. 5, we study the influence of grid refinement in the case of a 4-dimension problem.

First, according to Fig. 5a, we can see that the speed-up exceeds 5 in most cases of 4-dimension problems. It rises, using the whole GPU capacity but, then, the curve stagnates. The speed-up reaches a threshold when the GPU is fully loaded and where no more time gain is possible. On Fig. 5b, a comparison of elapsed times during the CPU and GPU version of the algorithm is shown. We notice that when the refinement of the grid is high, the computation time increases significantly in the CPU version.

Fig. 6, shows speed-up isolines in function of the number of dimensions and the number of points by dimension. We observe that when we have a 2- or 3-dimension problem, the speed-up is between 0 and 4. This is not significantly time-saving, and the initialization of the GPU can be slower than the classic array filling. However, when the number of dimensions grows, the speed-up increases up until 20 in 7 dimensions or more in 5 or 6 dimensions with a finer discretization. The speed-up isolevels follow the total number of points of the grid for high-dimension problems (4 dimensions or more). In these problems,

the main part of the time is spent in the trajectories storing part (the iterating kernel approximation ends quickly because of the low accuracy of the grid).

#### 4.1.3 Parallelized parts

Here, we compare the speed-up during the storing of successors and during the computation one between the CPU and the GPU versions of the algorithm. In the following figure (Fig. 7), we study two different sizes of problems (2 and 6 dimensions) and we return speed-ups in function of the number of points by dimension. Elapsed times during the two parallelized parts are also returned.

On Fig. 7a, we see that the elapsed time during the iterating kernel approximation is now higher than the one for storing the successors for 2-dimensions problems (for problems with more than 3000 points per dimension for the CPU version, or 1600 points per dimension for the GPU version). For problems in 3 dimensions or more (like 6 dimensions in Fig7c), storing successors time is still higher than iterating kernel approximation time. The time for storing successors exponentially increases with the number of dimensions and parallelizing this part of the algorithm is very efficient.

The speed-ups of the storing of trajectories and iterating kernel approximation are calculated separately here. As shown on Fig.7b and Fig.7d the speed-up increases up to 27 in 2 dimensions and 34 in 6 dimensions for the storing of trajectories, and up to 15 in 2 dimensions and 3.7 in 6 dimensions for the iterating kernel approximation.

For a small problem with a low discretization, the time taken by the two algorithms is negligible, but we observe that the speed-up is below one (up to 400 points by dimension in the 2D-problem, and 11 points by dimension in the 6D-problem). The mandatory initialization of the CUDA device explains the less successful results of the GPU version in smaller problems.



331 Moreover, we notice that the parallelization of the storing of successors is  
 332 more efficient whatever the number of dimensions. Actually, memory exchanges  
 333 (reading the value function array in the global memory from each thread at  
 334 each step) in the second part alter the performance of the parallelization.  
 335 Thus, the speed-up of the iterating kernel approximation falls down to 3.7 in  
 336 6 dimensions. Since the time elapsed during this step is small compared to the  
 337 time elapsed during the storing of trajectories, its impact is not significant on  
 338 the global speed-up.

## 339 4.2 Application to a bycatch fishery

340 We illustrate our results on a multidimensional bycatch fishery model. This  
 341 model is an extension of the one studied by Krawczyk et al (2013) from 3  
 342 dimensions to 6 dimensions. We study the influence of the number of bycatch  
 343 species on the fishery management.

### 344 4.2.1 *The two-species dynamics*

345 The model of Krawczyk is reminded here. There are two ecologically inde-  
 346 pendent populations of fish  $x$  and  $y$ , harvested by a single fleet. The fishery  
 347 focuses on harvesting the target stock  $x$ . We define a catchability coefficient  $q_x$   
 348 determining the quantity of biomass that each unit of effort extracts, relative  
 349 to the total size of the biomass at the time. Thus, the harvest rate at time  $t$   
 350 is:

$$h_x(t) = q_x e(t) x(t) \quad (10)$$

351 The product of  $q_x$  and  $e(t)$  is generally termed the level of fishing mortality  
 352 and it is expressed as a proportion of biomass.

353 A linear bycatch production function is used :

$$h_y(t) = \alpha h_x(t) \quad (11)$$

354 where the constant  $\alpha$ ,  $0 < \alpha < 1$  is the bycatch to target harvest ratio.

355 The respective population dynamics are then described by :

$$x(t+1) = x(t) + r_x x(t) \left(1 - \frac{x(t)}{L_x}\right) - q_x e(t) x(t) \quad (12)$$

$$y(t+1) = y(t) + r_y y(t) \left(1 - \frac{y(t)}{L_y}\right) - \alpha q_x e(t) x(t) \quad (13)$$

356 where  $r_x$ ,  $r_y$ ,  $L_x$  and  $L_y$  are all positive constants. By convention,  $r_x$  and  
 357  $r_y$  represent the intrinsic growth rates, and  $L_x$  and  $L_y$  are the environment  
 358 carrying capacities, of  $x$  and  $y$ , respectively. According to Krawczyk, the nu-  
 359 merical values for the bycatch species are chosen to have a bycatch stock less  
 360 productive and less valuable than the target stock.

361 We suppose that the effort  $e$  can be controlled:

$$362 \quad e(t+1) = e(t) + u(t), \quad u(t) \in [u_{min}, u_{max}]$$

363 The speed at which the regulator can change fishing intensity is bounded  
 364 by  $u_{min}$  and  $u_{max}$ . The fishing fleet's profit is given by :

$$\pi_{xy}(t) = p_x h_x(t) + p_y h_y(t) - ce(t) - C \quad (14)$$

365 where  $p_y$  represents the unit prices of  $y$ ,  $p_x$  the unit price of  $x$  The marginal  
 366 cost of the fishing effort is  $c$  and  $C$  is a fixed cost. We define the viability  
 367 constraints set  $K$  :

$$K = \left\{ (x, y, e, u) : x(t) \geq \frac{L_x}{10}, y(t) \geq \frac{L_y}{10}, \pi_{xy}(t) \geq 0, e(t) \in [e_{min}, e_{max}] \right\} \quad (15)$$

368 The safe minimum biomass levels are set to one tenth of each fish pop-  
 369 ulation's unexploited level, as commonly implemented in fisheries worldwide  
 370 (Krawczyk et al, 2013).

#### 371 4.2.2 The $n$ -species dynamics

372 We propose to extend this model to treat  $n$ -species dynamics in order to il-  
 373 lustrate the benefits of the GPU computation of the viability kernel. The  
 374 dynamics of the target species remains unchanged (Eq.2). We suppose here  
 375 that we have  $n$  bycatch species  $y_i$ . Moreover, we assume that these species are  
 376 at the same trophic level, competitively sharing the same food resource (Eker-  
 377 hovd and Steinshamm, 2014; Rice et al, 2013). The bycatch species growth is  
 378 limited by the common carrying capacity  $L_y$ . The discrete dynamics becomes:

$$y_i(t+1) = y_i(t) + r_{y_i} y_i(t) \left( 1 - \frac{\sum_{k=1}^{n-1} y_k(t)}{L_y} \right) - \alpha_i q_x e(t) x(t), \forall i \in [1, n-1] \quad (16)$$

379  $r_{y_i}$  is the intrinsic growth rate.  $\alpha_i$  denotes the bycatch to target harvest  
 380 ratio, a measure of how highly coupled the production relationship is. Each  
 381 bycatch stock  $y_i$  is a by-product of the production process. A linear bycatch  
 382 production function is used :

$$h_{y_i}(t) = \alpha_i h_x(t) \quad (17)$$

383 We assume here that the sum of all bycatch harvest ratios  $\alpha_i$  is equal to  $\alpha$   
 384 defined in the previous section :

$$\sum_{i=1}^{n-1} \alpha_i(t) = \alpha \quad (18)$$

385 For reasons of simplicity, we suppose that all the  $\alpha_i$  are equal :

$$\alpha_i = \frac{\alpha}{n} \quad (19)$$

386 As before, economic sustainability requires that the activity remains prof-  
 387 itable. The fishing fleet's profit is given by :

$$\pi_{xy}(t) = p_x h_x(t) + \sum_{i=1}^{n-1} p_{y_i} h_{y_i}(t) - ce(t) - C \quad (20)$$

388 Then, we have the viability constraint set  $K$  :

$$K = \left\{ (x, y_1, \dots, y_{n-1}, e, u) : x(t) \geq \frac{L_x}{10}, \forall i \in [1, n-1], y_i(t) \geq \frac{L_{y_i}}{10n}, \pi_{xy}(t) \geq 0, e(t) \in [e_{min}, e_{max}] \right\} \quad (21)$$

389 In the following, we study dynamics with 1, 2, 3 or 4 bycatch species, which  
 390 means dynamics with 3, 4, 5 or 6 dimensions respectively (including the fishing  
 391 effort and the target species biomass).

#### 392 4.2.3 Results

393 We use the parameters calibrated by Krawczyk (Krawczyk et al, 2013) and  
 394 summed up in the Table 1.

395 With only one bycatch species, a similar shape of the viability kernel is  
 396 obtained than the one in the previous study (Krawczyk et al, 2013). This

397 viability kernel is presented in Fig.8. For the following figures, the biomasses  
398 are scaled between 0.1 and 1.

399 The bounds of the figure are the constraint domain  $K$ . From each state  
400 contained inside the viability kernel, it exists at least a control strategy keeping  
401 the fishery in  $K$ . The states outside of the viability kernel are not viable, there  
402 is no control strategy satisfying the bio-economic constraints. For these states,  
403 the fishery is facing a "crisis" situation.

404 Now, bycatch species are added to the environment. For one bycatch species,  
405 41.9% of states are inside the viability kernel. It decreases until less than 7%  
406 for four bycatch species.

407 The Fig.9 shows 3D slices of the 4D kernel for different fishing effort val-  
408 ues. It represents the biomass of the target species and the biomass of the  
409 bycatch species required to be viable to satisfy the ecological constraints while  
410 maintaining the profitability of the fishery. A small fishing effort needs big  
411 available stock of the target biomass in order to conserve the system prof-  
412 itability. A mean fishing effort leads to a large slice of the viability kernel  
413 according to the target biomass. Finally, keeping a big fishing effort requires a  
414 mean target species stock to not deplete the bycatch populations. The results  
415 are displayed for 4D problems, but they can be extended to higher dimension  
416 problems.

417 Finally, we show on Fig.10 the number of states belonging to the viability  
418 kernel for a 6D problem with a grid of 20 points by dimension. The biggest  
419 viability kernel is obtained when the starting population of bycatch species are  
420 equal to  $\frac{L_y}{n}$  (the offset is set by the grid step which is nearly 15.3 here). The  
421 environment is then completely filled with balanced increasing the possibility  
422 of sustainability. For larger initial populations, the environment is overloaded,  
423 causing the depletion of bycatch species. When the initial populations are

424 smaller, the ecological sustainability requirement restricts the number of viable  
425 states.

## 426 **5 Discussion and conclusion**

427 The GPU parallelization provides significantly faster viability kernel computa-  
428 tions and tackles problems with a higher number of dimensions or more precise  
429 discretization than a sequential algorithm.

430 However, a huge storage space is required to save all the coordinates and all  
431 the successors for high-dimension dynamical problems. For example, in C++,  
432 storing one coordinate in double-precision floating-point format required 8  
433 bytes (64 bits). Then, storing the coordinates of 100 millions points in a 8-  
434 dimension problem (10 points per dimension) requires 6.4Gb. Moreover the  
435 successors multiplied this value by the number of possible values of the control.  
436 Because of this size, it is impossible to send all the data at once. Instead, we  
437 cut the problem in smaller parts : computing the coordinates for some grid  
438 points and then computing the successors  $g(x, u)$  for these points, the result  
439 is then saved. The process then continues with other grid points until all the  
440 whole grid has been processed. This reduces the need for memory.

441 This parallelized version may be improved with other tools (like STXXL  
442 in C++ to use large arrays) and using the mapped memory, which is based  
443 on some pointers to the RAM. Further improvements could be achieved by  
444 associating multi-CPU and multi-GPU architectures, the first ones cutting  
445 the problem in small parts and the second ones solving the subproblems.

446 The parallelization approach proposed here opens up perspectives in terms  
447 of parallelization for other viability problems. Then, revisiting other viabil-  
448 ity approaches such as an extension to stochastic dynamics (Doyen and De  
449 Lara, 2010) or algorithms such as support vector machine algorithm (Deffuant

450 et al, 2007) could also lead to significant improvements of efficiency. Finally,  
451 the parallelization of the viability kernel algorithm provides a tool for the  
452 management of socio-ecosystems as illustrated with the management of the  
453 multi-species bycatch fishery.

#### 454 **Acknowledgments**

455 This work was supported by grants from Irstea and Région Auvergne. This  
456 support is gratefully acknowledged.

#### 457 **References**

- 458 Andrés-Domenech P, Saint-Pierre P, Smala Fanokoa P, Zaccour G (2014)  
459 Sustainability of the Dry Forest in Androy: A Viability Analysis. *Ecologi-*  
460 *cal Economics* 104:33–49, DOI 10.1016/j.ecolecon.2014.04.016, URL <http://linkinghub.elsevier.com/retrieve/pii/S0921800914001256>
- 462 Aubin JP, Saint-Pierre P (2007) An Introduction to Viability Theory and  
463 Management of Renewable Resources. In: *Advanced Methods for Decision*  
464 *Making and Risk Management in Sustainability Science*, Nova Science Pub-  
465 *lishers, Inc*, pp 43–93
- 466 Béné C, Doyen L, Gabay D (2001) A viability analysis for a  
467 bio-economic model. *Ecological Economics* 36(3):385–396, DOI 10.  
468 1016/S0921-8009(00)00261-5, URL [http://linkinghub.elsevier.com/](http://linkinghub.elsevier.com/retrieve/pii/S0921800900002615)  
469 [retrieve/pii/S0921800900002615](http://linkinghub.elsevier.com/retrieve/pii/S0921800900002615)
- 470 Bernard C, Martin S (2013) Comparing the sustainability of different ac-  
471 tion policy possibilities: application to the issue of both household sur-  
472 vival and forest preservation in the corridor of Fianarantsoa. *Mathemati-*  
473 *cal biosciences* 245(2):322–30, DOI 10.1016/j.mbs.2013.08.002, URL [http://](http://www.ncbi.nlm.nih.gov/pubmed/23954403)  
474 [www.ncbi.nlm.nih.gov/pubmed/23954403](http://www.ncbi.nlm.nih.gov/pubmed/23954403)

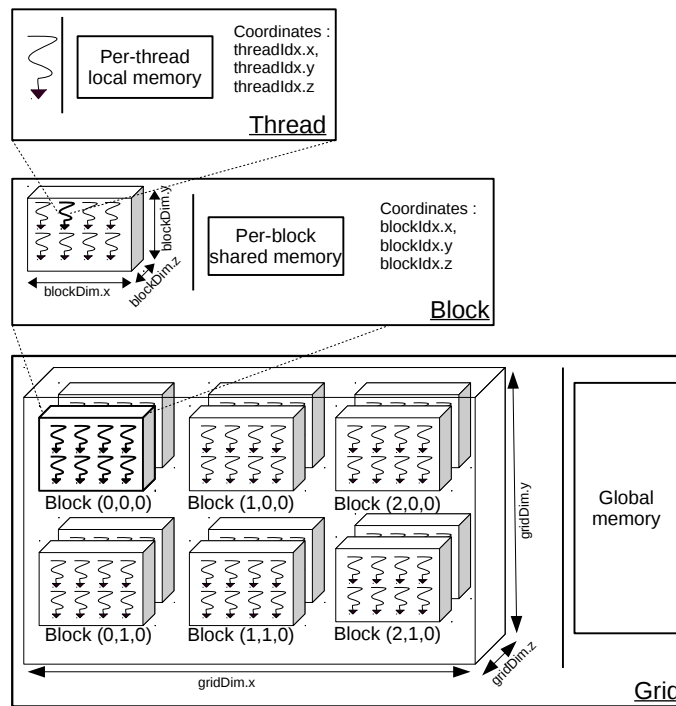
- 475 Bokanowski O, Martin S, Munos R, Zidani H (2006) An anti-diffusive scheme  
476 for viability problems. *Applied Numerical Mathematics* 56(9):1147–1162,  
477 DOI 10.1016/j.apnum.2006.03.004, URL [http://linkinghub.elsevier.  
478 com/retrieve/pii/S0168927406000419](http://linkinghub.elsevier.com/retrieve/pii/S0168927406000419)
- 479 Bonneuil N (2006) Computing the viability kernel in large state dimen-  
480 sion. *Journal of Mathematical Analysis and Applications* 323(2):1444–1454,  
481 DOI 10.1016/j.jmaa.2005.11.076, URL [http://linkinghub.elsevier.  
482 com/retrieve/pii/S0022247X05012916](http://linkinghub.elsevier.com/retrieve/pii/S0022247X05012916)
- 483 Cekmez U, Ozsiginan M, Technologies S, Air T, Academy F, Sahingoz OK  
484 (2013) Adapting the GA Approach to Solve Traveling Salesman Problems  
485 on CUDA Architecture. In: 14th IEEE International Symposium on Com-  
486 putational Intelligence and Informatics, pp 423–428
- 487 Chapel L, Deffuant G, Martin S, Mullon C (2008) Defining yield poli-  
488 cies in a viability approach. *Ecological Modelling* 212(1-2):10–15, DOI 10.  
489 1016/j.ecolmodel.2007.10.007, URL [http://linkinghub.elsevier.com/  
490 retrieve/pii/S030438000700525X](http://linkinghub.elsevier.com/retrieve/pii/S030438000700525X)
- 491 Chapel L, Castelló X, Bernard C, Deffuant G, Eguíluz VM, Martin S, San  
492 Miguel M (2010) Viability and resilience of languages in competition. *PLoS*  
493 *ONE* 5(1):1–11, DOI 10.1371/journal.pone.0008681
- 494 Deffuant G, Chapel L, Martin S (2007) Approximating Viability Kernels  
495 With Support Vector Machines. *IEEE Transactions on Automatic Control*  
496 52(5):933–937, DOI 10.1109/TAC.2007.895881, URL [http://ieeexplore.  
497 ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4200855](http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4200855)
- 498 Djeridane B, Lygeros J (2008) Approximate Viability using Quasi-Random  
499 Samples and a Neural Network Classifier. In: *The International Federation*  
500 *of Automatic Control*, 2, pp 14,342–14,347
- 501 Donoho DL (2000) High-Dimensional Data Analysis: The Curses and Blessings  
502 of Dimensionality. In: *Math Challenges of the 21st Century*, pp 1–32



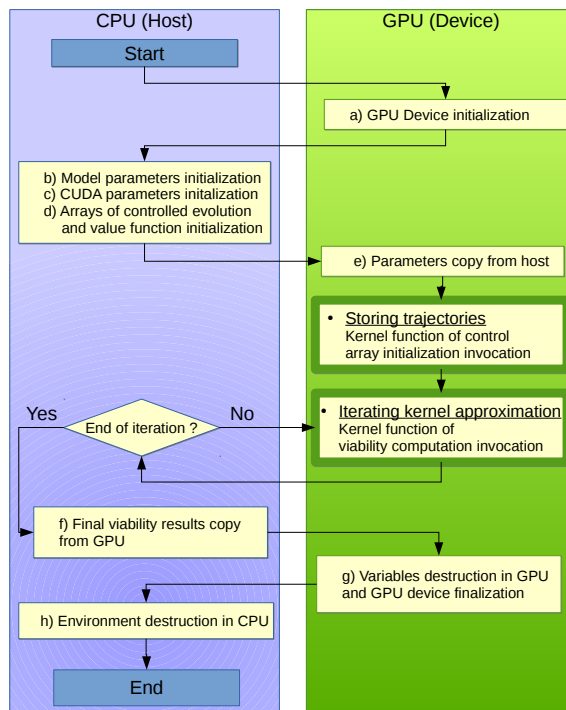
- 503 Doyen L, De Lara M (2010) Stochastic viability and dynamic pro-  
504 gramming. *Systems & Control Letters* 59(10):629–634, DOI 10.1016/j.  
505 sysconle.2010.07.008, URL [http://linkinghub.elsevier.com/retrieve/  
506 pii/S0167691110000952](http://linkinghub.elsevier.com/retrieve/pii/S0167691110000952)
- 507 Doyen L, Thébaud O, Béné C, Martinet V, Gourguet S, Bertignac M, Fi-  
508 fas S, Blanchard F (2012) A stochastic viability approach to ecosystem-  
509 based fisheries management. *Ecological Economics* 75:32–42, DOI 10.1016/j.  
510 ecolecon.2012.01.005, URL [http://linkinghub.elsevier.com/retrieve/  
511 pii/S0921800912000067](http://linkinghub.elsevier.com/retrieve/pii/S0921800912000067)
- 512 Ehrlich I, Lui F (1997) The problem of population and growth: a review of the  
513 literature from Malthus to contemporary models of endogenous population  
514 and endogenous growth. *Journal of economic dynamics & control* 21(1):205–  
515 42, URL <http://www.ncbi.nlm.nih.gov/pubmed/12292267>
- 516 Ekerhovd NA, Steinshamm SI (2014) Optimization in the ‘Pelagic Complex’:  
517 A Multi-Species Competition Model of North East Atlantic Fisheries
- 518 Goldsworthy MJ (2014) A GPU–CUDA based direct simulation Monte Carlo  
519 algorithm for real gas flows. *Computers & Fluids* 94:58–68, DOI 10.  
520 1016/j.compfluid.2014.01.033, URL [http://linkinghub.elsevier.com/  
521 retrieve/pii/S0045793014000516](http://linkinghub.elsevier.com/retrieve/pii/S0045793014000516)
- 522 Kaynama S, Oishi M (2013) A Modified Riccati Transformation for Decen-  
523 tralized Computation of the Viability Kernel Under LTI Dynamics. *IEEE*  
524 *Transactions on Automatic Control* 58(11):2878–2892, arXiv:1302.5990v1
- 525 Krawczyk JB, Pharo A, Serea OS, Sinclair S (2013) Computation of via-  
526 bility kernels: a case study of by-catch fisheries. *Computational Manage-*  
527 *ment Science* 10(4):365–396, DOI 10.1007/s10287-013-0189-z, URL [http:  
528 //link.springer.com/10.1007/s10287-013-0189-z](http://link.springer.com/10.1007/s10287-013-0189-z)
- 529 Langdon WB (2011) Debugging CUDA. *Proceedings of the 13th annual con-*  
530 *ference companion on Genetic and evolutionary computation - GECCO*

- 531 '11 p 415, DOI 10.1145/2001858.2002028, URL [http://portal.acm.org/  
532 citation.cfm?doid=2001858.2002028](http://portal.acm.org/citation.cfm?doid=2001858.2002028)
- 533 Maidens JN, Kaynama S, Mitchell IM, Oishi MMK, Dumont GA (2009) La-  
534 grangian methods for approximating the viability kernel in high-dimensional  
535 systems. *Automatica* (2013) 49(7):2017–2029
- 536 Mametjanov A, Lowell D, Ma CC, Norris B (2012) Autotuning Stencil-Based  
537 Computations on GPUs. 2012 IEEE International Conference on Cluster  
538 Computing pp 266–274, DOI 10.1109/CLUSTER.2012.46, URL [http://  
539 ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6337788](http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6337788)
- 540 Mathias JD, Bonté B, Cordonnier T, de Morogues F (2015) Using the  
541 Viability Theory to Assess the Flexibility of Forest Managers Under  
542 Ecological Intensification. *Environmental Management* (JUNE), DOI  
543 10.1007/s00267-015-0555-4, URL [http://link.springer.com/10.1007/  
544 s00267-015-0555-4](http://link.springer.com/10.1007/s00267-015-0555-4)
- 545 Mattioli J, Artiouchine K (2003) Noyau de viabilité : une contrainte globale  
546 pour la modélisation de systèmes dynamiques. *Technique et Science Infor-  
547 matiques* pp 19–32
- 548 Rice J, Daan N, Gislason H, Pope J (2013) Does functional redundancy sta-  
549 bilized fish communities 70:734–742
- 550 Rougé C, Mathias JD, Deffuant G (2013) Extending the viability the-  
551 ory framework of resilience to uncertain dynamics, and application to  
552 lake eutrophication. *Ecological Indicators* 29:420–433, DOI 10.1016/j.  
553 ecolind.2012.12.032, URL [http://linkinghub.elsevier.com/retrieve/  
554 pii/S1470160X13000204](http://linkinghub.elsevier.com/retrieve/pii/S1470160X13000204)
- 555 Sabo D, Barzelay O, Weiss S, Furst M (2014) Fast evaluation of a  
556 time-domain non-linear cochlear model on GPUs. *Journal of Computa-  
557 tional Physics* 265:97–112, DOI 10.1016/j.jcp.2014.01.044, URL [http://  
558 linkinghub.elsevier.com/retrieve/pii/S0021999114000898](http://linkinghub.elsevier.com/retrieve/pii/S0021999114000898)

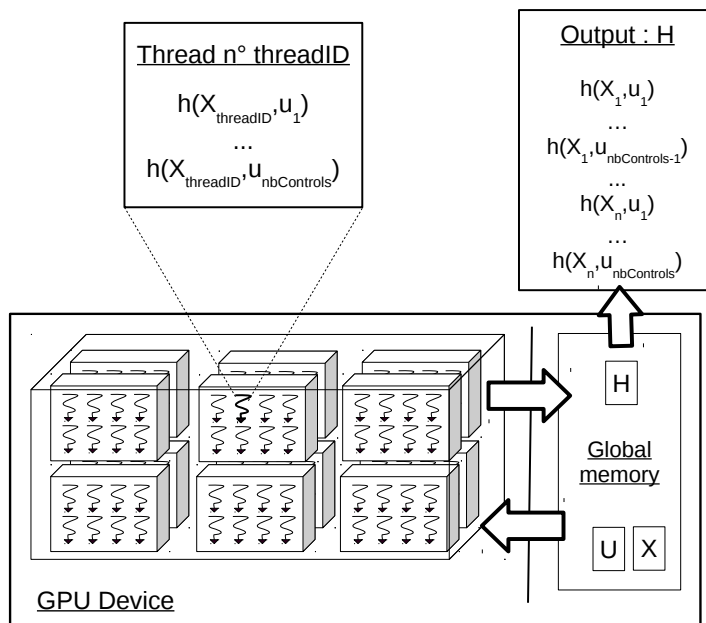
- 
- 559 Saint-Pierre P (1990) Approximation of the Viability Kernel. *Applied Mathe-*  
560 *matics & Optimization* 29(2):1–22
- 561 Sicard M, Perrot N, Reuillon R, Mesmoudi S, Alvarez I, Martin S  
562 (2012) A viability approach to control food processes: Application to a  
563 Camembert cheese ripening process. *Food Control* 23(2):312–319, DOI  
564 10.1016/j.foodcont.2011.07.007, URL [http://linkinghub.elsevier.com/  
565 retrieve/pii/S0956713511002805](http://linkinghub.elsevier.com/retrieve/pii/S0956713511002805)
- 566 Turriff J, Broucke ME (2009) A method to construct viability kernels for  
567 nonlinear control systems. *2009 American Control Conference* 1(1):3983–  
568 3988, DOI 10.1109/ACC.2009.5160409, URL [http://ieeexplore.ieee.  
569 org/lpdocs/epic03/wrapper.htm?arnumber=5160409](http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5160409)



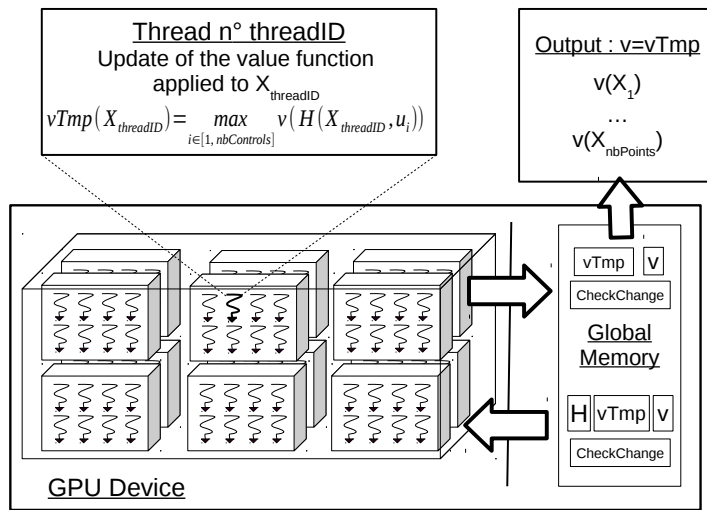
**Fig. 1** GPU device architecture. The device is composed of a grid of threads blocks. Each level possesses its own memory. The parallelized tasks, like computation of controlled evolutions and computations of the value function, are made separately on the threads. A grid of  $3 \times 2 \times 2$  blocks of  $4 \times 2 \times 1$  threads is shown.



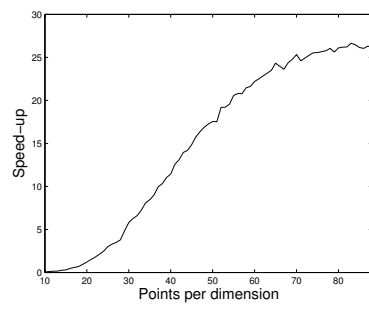
**Fig. 2** Organization of the algorithm. Steps a,b,c,d,e,f,g and h are explained in 3.2. Kernel functions are detailed in 3.3 and in 3.4.



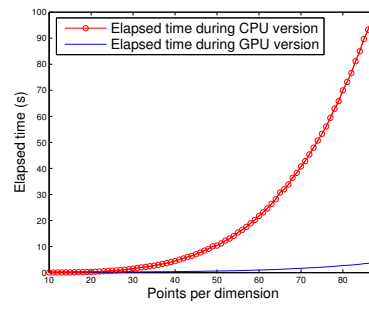
**Fig. 3** Storing of trajectories : the parallelized controlled evolution computation from coordinates. The different steps of the parallelization are shown. The computation of all the controlled evolution for one point is done on one thread.



**Fig. 4** Iterating kernel approximation: the parallelized viability computation. The different steps of the parallelization are shown.



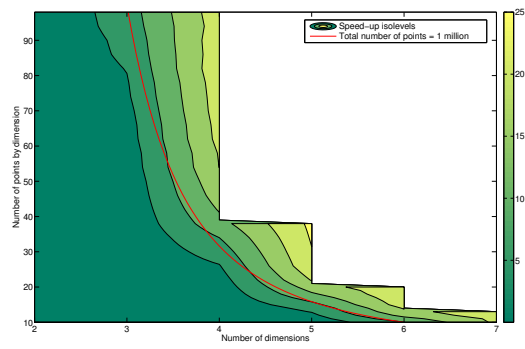
(a) Speed-up



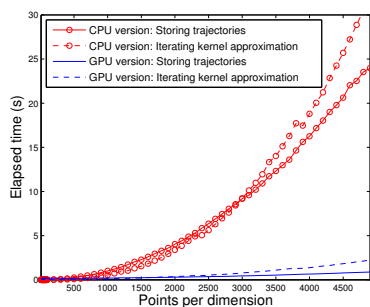
(b) Comparison of elapsed times

**Fig. 5** Effect of grid refinement on the speed-up in a 4-dimensions problem. The elapsed time during the two versions of the algorithm are shown in (b).

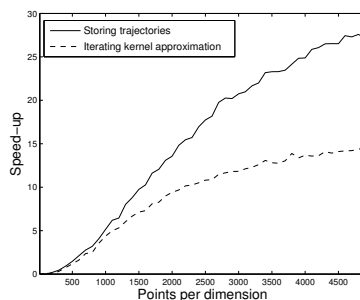




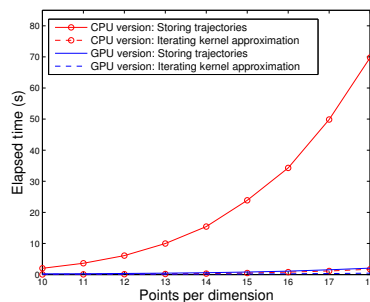
**Fig. 6** Speed-up isolevels. The red line shows problems with same amount of points on the grid (1 million).



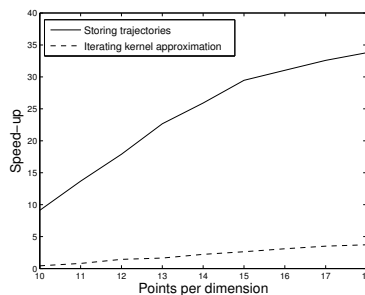
(a) Comparison of elapsed times (2D)



(b) Speed-up of the storing of trajectories and iterating kernel approximation (2D)

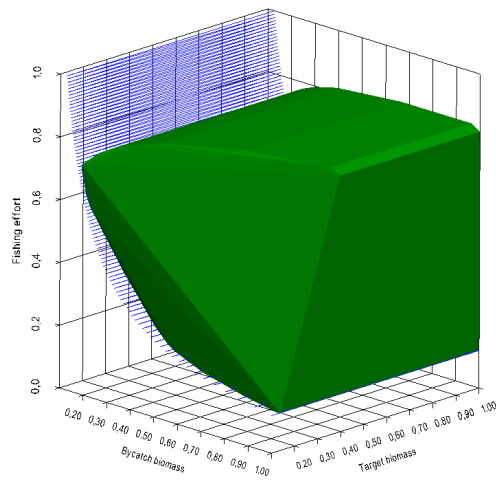


(c) Comparison of elapsed times (6D)

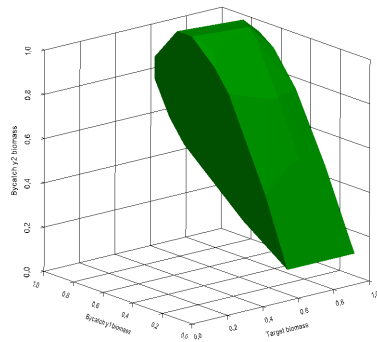
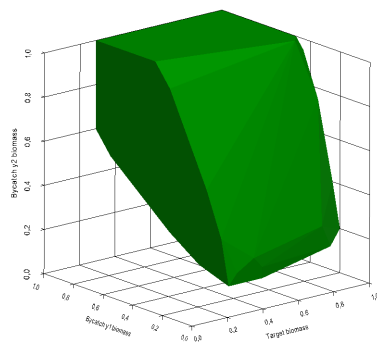
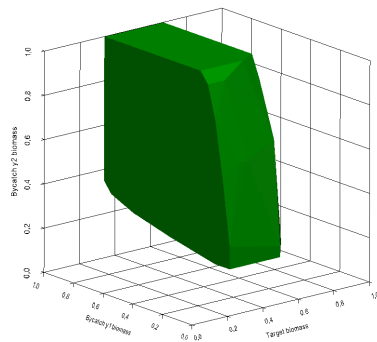


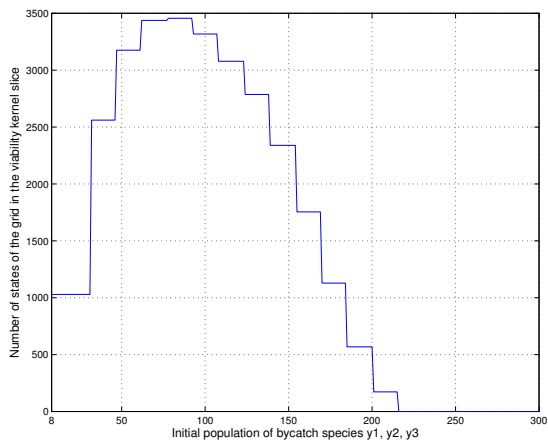
(d) Speed-up of the storing of trajectories and iterating kernel approximation (6D)

**Fig. 7** Comparison of the storing of trajectories and the iterating kernel approximation elapsed times and speed-up between the GPU and CPU versions of the algorithm. Two sizes of problems are shown : 2-dimension problem (a) and (b), and 6-dimension problem (c) and (d).



**Fig. 8** Viability kernel for the 3D fishery model (with one bycatch species). Viability kernel for the 3D fishery model (with one bycatch species). The blue surface symbolizes the zero-profit surface. Having a positive profit (being above the blue surface) is a property of the viability constraint set  $K$ .

(a) Slice through  $e = 0.2$ (b) Slice through  $e = 0.5$ (c) Slice through  $e = 0.8$ **Fig. 9** 3D slices of the 4D kernel for different initial values of the fishing effort  $e$ .



**Fig. 10** Number of states of the grid belonging to the viability kernel for a 6D problem. The starting populations of three bycatch species are equal.

Name	Variable	Value
Target carrying capacity	$L_x$	600
Target growth rate	$r_x$	0.4
Target unit price	$p_x$	4
Target catchability coefficient	$q_x$	0.5
Bycatch common carrying capacity	$L_y$	300
Bycatch growth rate	$r_{y_i}$	0.2
Bycatch unit price	$p_{y_i}$	1.9
Bycatch harvest ratio	$\alpha_{y_i}$	0.2
Marginal cost	$c$	10
Fixed cost	$C$	150
Maximum effort	$e_{max}$	1
Minimum effort	$e_{min}$	0.1
Maximum effort variation	$u_{max}$	0.01
Minimum effort variation	$u_{min}$	-0.01

**Table 1** Parameters of the bycatch fishery. The parameters for all the bycatch species  $i$  are the same.