22nd International Congress on Modelling and Simulation, Hobart, Tasmania, Australia, 3 to 8 December 2017
mssanz.org.au/modsim2017

Using GPUs to improve computation time of optimal road design in ecologically-sensitive areas

N. Davey [a], **S. Dunstall** [b] **and S. Halgamuge** [c]

[a]*Melbourne School of Engineering, The University of Melbourne, Parkville, VIC 3010, Australia*
[b]*CSIRO Data 61, Docklands, VIC 3008, Australia*
[c]*The Australian National University, Acton, ACT 2601, Australia*
Email: ndavey@student.unimelb.edu.au

Abstract: Heavy vehicle movements along mining haul roads have significant negative impacts on the migration behaviour and survival rates of nearby animal species. As the extinction of certain animal species can have dire consequences for the ecosystem and an entity's license to operate, mine operators need to consider these negative effects at the design and selection stage of new road projects. This necessitates the development of computationally-efficient techniques to optimise the preliminary design of new roads in the presence of ecological uncertainty.

Due to the presence of multiple local optima and non-linear objectives and constraints, the road design problem is typically solved using population-based search techniques such as Genetic Algorithms. Additionally, computing the operating value over the design life of the road requires using Optimal Control (in the form of real options valuation) methods that account for endogenous uncertainty and the ability to alter traffic flow rates. Finally, the evaluation process requires computing the habitat patches and simulating the migration of animals at each step of the optimal control algorithm. Individually, each of these techniques can be computationally challenging but together, they can render even a simple road optimisation through an ecologically-sensitive region intractable.

Existing research has addressed some of the speed limitations through the use of surrogate modelling and dimensionality-reduction techniques. In this paper, we build upon this research by implementing the most computationally-intensive components of the road design process to run on Graphics Processing Units (GPUs) and develop algorithms that take into account the highly-parallel nature of algorithm components (such as Monte Carlo simulation) in the road design optimisation process. In particular, we discuss the implementation of GPUs for three main problem components:

1. A road crossings algorithm to determine the number of road crossings between every pair of habitat patches for which there is a possible transition. This is used to compute the survival probabilities for the patch transitions.
2. A multiple local linear regression that is used at each time step of the optimal control sub-problem to compute the conditional expectations under each available control.
3. A real-options/optimal control scheme with forward path recomputation that incorporates the animal movement and mortality model. This is used to compute the value of the road (with inbuilt ability to alter the traffic flow) in the presence of animal population growth rate uncertainty and commodity price uncertainty.

We show that the speed improvement achieved from implementing these techniques can dramatically increase the ability of road designers to evaluate many alternative options, thus enabling a global search for high value roads in complex regions. We demonstrate this using an example of a typical road encountered when running a global search routine to find a high value road. The improvements presented in this work can be extended to other research using optimal control to value complex systems with endogenous uncertainty.

Keywords: *GPU programming, road design, real options valuation, Monte Carlo simulation, ecological model*

1 INTRODUCTION

Optimal road design involves finding high-value roads connecting two points through a terrain so as to minimise journey times or maximise throughput (Kang et al., 2012). Additionally, there is a need to optimally control the traffic flow over such roads when they pass through regions containing vulnerable animal species so that minimum animal population requirements are maintained (Davey et al., 2017).

This problem is subject to multiple local optima and discontinuous and non-linear costs and constraints (Jong and Schonfeld, 2003). For this reason, population-based global search algorithms are popular approaches (Jha et al., 2006). Another advantage of these methods is that they evaluate entire candidate roads, which is important in dealing with the spatial connectivity of animal species when computing operational flexibility. This flexibility component evaluates the operating value of a candidate road where operators can optimally alter traffic flow in response to exogenous (fuel and commodity prices) and endogenous (population growth) uncertainties using stochastic optimal control based on Monte Carlo simulation (Kharroubi et al., 2014; Langrené et al., 2015). The reader is encouraged to consult Davey et al. (2017) and Davey et al. (2016), which describe the road evaluation components in greater detail.

Unfortunately, the three main components used in this process (A. the global search algorithm, B. the optimal control algorithm and, C. the spatial ecological model) are computationally-demanding. This is exacerbated by the fact that each candidate road generated by the global search algorithm (component A) is evaluated by the optimal control algorithm (component B), which in turn implements the spatial ecological model (component C) (Davey et al., 2018). Due to the nesting of each algorithm within the next, the road design problem can quickly become intractable. Davey et al. (2017, 2018) addressed some of this burden through the use of surrogate functions to avoid computing the full optimal control algorithm for every candidate road. Instead, their methods strategically sample a smaller number of roads at each generation on which to compute the full optimal control model. The results from these samples are then used to update a much faster surrogate function to approximate the operating costs of the remaining roads. The surrogate uses the number of animals expected to perish and the unit operating costs in the first time period as predictors of the operating value.

Even with the use of surrogate functions, the full optimal control model still needs to be carried out for many candidate roads (albeit far fewer than in the absence of the surrogate). This is problematic as ecological uncertainty is endogenous, which requires recomputation of the Monte Carlo paths at each time step of the backward induction (Langrené et al., 2015; Kharroubi et al., 2014). In addition, the conditional expectations at each time step are computed using local-linear regression, which requires implementing a k-nearest neighbour algorithm at each query point. Finally, the ecological model requires determining the number of road crossings for every patch transition in order to compute the initial animals at risk for a candidate road. When computed for hundreds of roads within a road optimisation algorithm, these aspects pose a major challenge, even for the most powerful of modern computers.

To reduce the computation time required, we use Graphical Processing Units (GPUs) to take advantage of the high levels of parallelism of three road design components. GPUs allow parallel processing of substantially more threads than CPUs (Nickolls et al., 2008), which makes them perfectly suited to applications involving Monte Carlo simulation (Lee et al., 2010; Sampathirao et al., 2015). In this paper, we present formulations of the algorithms for three key computationally-expensive components of the road design algorithm: 1. computing road crossings, 2. performing local linear regression, and 3. computing stochastic optimal control with forward path computation and ecological uncertainty. We also discuss how their GPU implementations can drastically improve computation times over their single-threaded versions.

2 GPU KERNELS

Table 1 shows the average contribution of every major component of the design and evaluation of a candidate road over a design life of 30 years. It shows that the three most computationally-demanding components when evaluating a single road are, in decreasing significance: path computations, regressions and the road crossings computation. For the case where the surrogate is used, the remaining components contribute approximately 20% of the computation time. These include animal patch computation and the calculation of other road costs components.

These three components require many parallel operations and are ideally suited for computation on GPUs. GPUs employ significanly more compute cores than standard CPUs and take advantage of low-latency on-chip memory to drastically improve computation times (Ryoo et al., 2008). Code is run on a GPU through user-defined functions called *kernels* (Nickolls et al., 2008), which describe the operations for a single thread.

Using Surrogate		**Full Model**	
Component	*Breakdown (%)*	*Component*	*Breakdown (%)*
Alignment generation	3.1	Alignment generation	∼0
Alignment evaluation		Alignment evaluation	
Road crossings computation	78.0	Road crossings computation	0.1
Surrogate evaluation	0.2	Full model evaluation	
		Regressions	2.6
		Path computations	97.3
Remaining components	18.7	Remaining components	∼0
Total	*100%*	Total	*100%*

Table 1. Relative computational burden of road design and evaluation components.

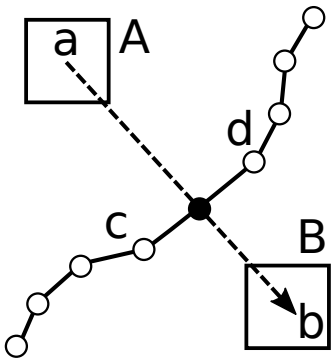

Figure 1. A transition across a segmented road

Algorithm 1. Line segment intersect kernel

1: **procedure** PATHINTERSECT(noSegments, **roadSegments**, **transitions**, **intersections**)
2: idx ← blockIdx × threadsPerBlock + threadIdx
3: seg ← ⌊idx/noSegments⌋
4: trans ← idx − seg × noSegments
5: **a** ← transitions[trans].start
6: **b** ← transitions[trans].end
7: **c** ← roadSegments[seg].start
8: **d** ← roadSegments[seg].end
9: $u \leftarrow (\mathbf{c}-\mathbf{a}) \times (\mathbf{b}-\mathbf{a})/((\mathbf{b}-\mathbf{a}) \times (\mathbf{d}-\mathbf{c}))$
10: **intersections**[idx] ← 0
11: **if** $(0 \leq u \leq 1)$ **then**
12: **intersections**[idx] ← 1
13: **end if**
14: **end procedure**

These kernels are called on thousands of threads at a time that operate on different parallel components of a problem. These threads are in turn grouped into thread blocks. Threads within the same block can also share on-chip memory to reduce redundant recomputation of code and multiple low-bandwidth accesses to high-latency global memory (Ryoo et al., 2008). Furthermore, each thread can be identified by its unique global index. This is based on its *block index* and *thread index* within its thread block. This index allows a thread to access the appropriate memory locations from within the kernel call (Nickolls et al., 2008).

The following subsections outline how the three different components of the road design problem can be structured so as to run on GPUs.

2.1 Road Crossings

While the Table 1 shows that the relative significance of the road crossings kernel is far lower than that of the optimal control components, its computational burden is still significantly higher than that of the remaining road design and evaluation components. More importantly, it needs to be computed for every road generated in the global search algorithm, irrespective of whether the surrogate model is used or not.

Figure 1 depicts a single transition (among a total of K transitions between all patches in the region for a given road) from one population patch, *A* to another patch, *B* that traverses the road. As a road is represented by a sequence of N straight line segments and can potentially be very windy, computing the number of road crossings for a single transition consists of finding how many of the road segments the transition line intersects. Therefore, computing the number of road crossings for all possible transitions consists of finding $K \times N$ straight line intersections.

Algorithm 1 shows the pseudocode for this simple kernel, showing how to use a thread's unique index to perform the calculation on the correct segment-transition pair. This kernel makes use of the vector cross product of the path $\mathbf{a} \rightarrow \mathbf{b}$ and straight line segment $\mathbf{c} \rightarrow \mathbf{d}$ shown in Figure 1.

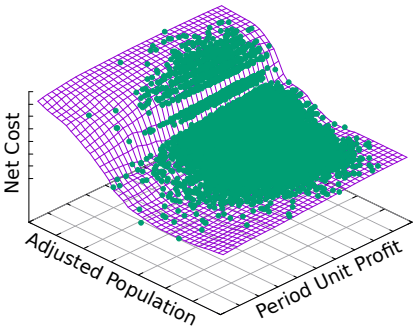

Figure 2. Non-parametric regression

Algorithm 2. Local linear regression kernel

1: **procedure** LOCALLINREG(k, **queryPoints**, **dataPoints**, **regression**)
2: idx ← blockIdx × threadsPerBlock + threadIdx
3: **neighbours** ← computeNearestNeighbours(k,**dataPoints**, **queryPoints**[idx])
4: **regression**[idx] ← computeRegression(**neighbours**, **queryPoints**[idx])
5: **end procedure**

The results are then summed over all road segments to give the number of crossings for each transition.

2.2 Multiple Local Linear Regression

At each step of the backward induction of the optimal control algorithm, the conditional expectations under each control are regressed against the predictor variables (in our case, an adjusted population for each animal species that takes into account spatial variability of the animals and the current period operating value (Davey et al., 2018)). Langrené et al. (2015) argued that a non-parametric approach such as local regression is well-suited to the Regression Monte Carlo method used to value the operational flexibility due to its ability to more faithfully fit typical payoff functions than traditional polynomial approaches such as that used in Longstaff and Schwartz (2001).

Consider the single period regression for the optimal control algorithm shown in Figure 2. To compute the regression value at each query point represented by the intersection points of the grid, we must first find the k nearest data points of each query point before performing a local regression. Given n data points, for each of the m query points, $\mathcal{O}\left(nlog(n)\right)$ operations need to be performed before computing the local regression in order to find the actual k nearest neighbours of each point.

The GPU kernel is invoked on the m query points and involves two key steps: 1. Find the k nearest neighbours to the point and 2. Perform the regression. This is shown by Algorithm 2. To improve computation time, the raw data points are loaded into on-chip memory for simultaneous use in all threads within a block. Further details on the use of on-chip memory can be found in Nickolls et al. (2008).

2.3 Optimal Control

As seen in Table 1, the optimal control algorithm is the most computationally-demanding and complex component of the road evaluation. Unlike the previous two algorithms discussed, the optimal control problem requires multiple calls to different kernels as summarised in Algorithm 3. Firstly, an initial paths kernel is called to launch one thread for each Monte Carlo path. This produces instances of the uncertain variables (fuel and commodity prices and population growth rates) and random controls (traffic flow rates) at each timestep (Langrené et al., 2015). Next, backward induction is performed at each time step to compute the conditional expectations and determine the optimal controls. This uses Algorithm 2. Finally, the forward paths are recomputed at each time step (Algorithm 4) using the recently-found optimal controls to account for the endogenous uncertainty of the animal populations. This last requirement effectively increases the number of time step calculations in the optimal control model from n (as is the case in traditional problems with only exogenous uncertainties) to $n(n+1)/2$. It is therefore the most computationally-demanding sub-process.

Algorithm 3 shows that the local linear regression and optimal forward path kernels are called multiple times for a single road evaluation. It also shows that when performing a kernel call, the number of blocks and threads per block must be provided to the GPU (delimited by $\langle\langle\langle\rangle\rangle\rangle$). The maximum number of threads per block is limited by the GPU hardware, with most later-model GPUs allowing a maximum of 1024 threads per block. Depending on on-chip memory availability and problem structure, it may be more optimal to have a lower number of threads per block (Nickolls et al., 2008).

As stated earlier, the most time-consuming component of the optimal control valuation is the forward path recomputation. This step requires recalculating the animal transitions between habitat patches at each time step

Algorithm 3. Optimal Control Valuation

```
1: procedure OPTIMALCONTROL
2:     noThreadsPerBlock ← maxThreadsPerBlock
3:     noBlocks ← ⌈noPaths/noThreadsPerBlock⌉
4:     ⟨⟨⟨noBlocks,noThreadsPerBlock⟩⟩⟩InitialPathsKernel(noPaths, noYears, transitionMatrix,
           initialPopulations, speciesPopulations, unitProfits, controls)
5:     for (noYears ≥ ii > 0) do
6:         for (0 ≤ jj < noControls) do
7:             queryPoints ← generateQueryPoints(speciesPopulations,unitProfits, controls)
8:             dataPoints ← generateDataPoints(speciesPopulations,unitProfits, controls)
9:             noBlocks ← ⌈queryPoints.size()/noThreadsPerBlock⌉
10:            ⟨⟨⟨noBlocks,noThreadsPerBlock⟩⟩⟩LocalLinReg(queryPoints, dataPoints, regressions[ii,jj])
11:        end for
12:        noBlocks ← ⌈noPaths/noThreadsPerBlock⌉
13:        ⟨⟨⟨noBlocks,noThreadsPerBlock⟩⟩⟩OptimalForwardPaths(speciesPopulations, unitProfits,
               regressions, conditionalExpectations, optimalControls, ii)
14:    end for
15:    for (0 ≤ ii < noControls) do
16:        expectedPayoffs[ii] ← averageFirstPeriodConditionalExpectation(conditionalExpectations,ii)
17:    end for
18:    ⟨⟨⟨noBlocks,noThreadsPerBlock⟩⟩⟩OptimalForwardPaths(speciesPopulations, unitProfits,
           regressions, conditionalExpectations, optimalControls, 0)
       return max(expectedPayoffs)
19: end procedure
```

after choosing the optimal control at each future time period based on the previously-computed conditional expectations. Like the initial generation of the Monte Carlo paths, the optimal forward paths kernel computes the movement of the animals throughout the patches from the current time period to the final but uses the optimal control instead of a random control.

Selection of the optimal control at each time period for each path requires determining an adjusted population (Davey et al., 2017) for each available control. This means computing a single period animal population transition for each control and the corresponding current period payoff to evaluate the conditional expectation. The optimal control is then the one with the maximum conditional expectation.

3 AN EXAMPLE ROAD

We ran the design and evaluation routines on the sample road shown in Figure 5. The corresponding habitat patches around this road are shown in Figure 6. The CPU implementation was written in C++ and run on an Intel Core i7 CPU @4.2GHz with 8GB of RAM while GPU code was written in CUDA C and run on an nVidia Quadro K5200, running 2,304 compute cores with 8GB video RAM. The simulation was performed over 50,000 Monte Carlo paths for 30 time steps with one endogenous uncertainty (animal population growth rates) and one exogenous uncertainty (fuel prices) and three control options (traffic flows). The results are shown in Table 2.

Overall, the GPU was able to reduce the computation times of the three algorithms discussed in this paper by a factor of approximately 22. For the case of evaluating the road using the surrogate model, this resulted in these components' contributions reducing from 81.3% of the total computation time to just 16%. In particular, a speedup of 173 was achieved for the road crossings. This is not unexpected as each thread for this component is simple and requires the use of only a few variables, thereby maximising the use of low-latency on-chip memory. The performance gains for the full model were not as high due to the more complex nature of the GPU kernels required. For example, each thread of the path recomputation kernel was required to update the animal populations in every patch at each time step. This meant a higher memory burden and more accesses to lower-bandwidth global memory. However, given that this component constitutes over 95% of the algorithm computation time, the performance improvements had the most significant benefit overall. As a whole, these are significant improvements that will dramatically increase the number of roads evaluated in a single generation of the overall road design algorithm.

It should be noted that further speed improvements can be made to the components not discussed in this paper. Most notably, the computation of habitat patches, described in Davey et al. (2017) can be improved by having

Algorithm 4. Optimal forward path kernel

```
1: procedure OPTIMALFORWARDPATHS(speciesPopulations, unitProfits, regressions, conditionalExpectations, optimalControls, startPeriod)
2:     idx ← blockIdx × threadsPerBlock + threadIdx
3:     for (0 ≤ ii < noControls) do
4:         adjustedPopulations ← computeAdjustedPopulations(speciesPopulations[startPeriod],ii)
5:         if adjustedPopulations ≥ minimimPopulation then
6:             payoffs[ii] ← unitProfits[idx,startPeriod]
7:         else
8:             payoffs[ii] ← NaN
9:         end if
10:    end for
11:    optimalControls[idx,startPeriod] ← argmax(payoffs)
12:    if (startPeriod == noYears) then
13:        conditionalExpectations[idx,startPeriod] ← max(payoffs)
14:    else
15:        conditionalExpectations[idx,startPeriod] ← max(payoffs)
16:        for (startPeriod < ii < noYears) do
17:            speciesPopulations[idx,ii] ← nextPeriodPopulations(speciesPopulations[idx,ii-1],
                   optimalControls[idx,ii-1])
18:            conditionalExpectations[idx,startPeriod] ← conditionalExpectations[idx,startPeriod] +
                   interpolateSurrogate(regressions[startPeriod,ii](adjustedPopulations, unitProfits[idx,ii]))
19:        end for
20:    end if
21:    neighbours ← computeNeighbours(k,dataPoints,
           queryPoints[idx])
22: end procedure
```

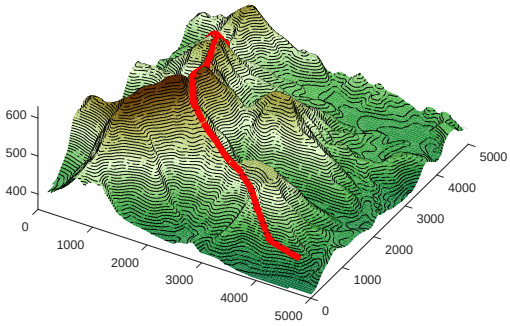

Figure 5. Example road over terrain

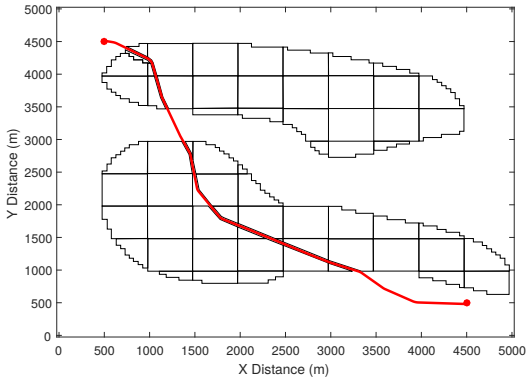

Figure 6. Corresponding habitat patches

Component	**CPU**				**GPU**			
	Surrogate		Full Model		Surrogate		Full Model	
	Time (s)	*%*	*Time (s)*	*%*	*Time (s)*	*%*	*Time (s)*	*%*
Alignment generation	0.046	3.1	0.046	∼0	0.046	13.1	0.046	0.1
Alignment evaluation								
Road crossings	1.211	78.0	1.211	0.1	0.007	2.0	0.007	∼0
Surrogate evaluation	0.003	0.2	–	–	0.003	0.9	–	–
Full model								
Regressions	–	–	33.3	2.6	–	–	2.16	3.1
Path computations	–	–	1,242.0	97.3	–	–	66.3	96.4
Remaining components	0.295	18.7	0.295	∼0	0.295	84.0	0.295	0.4
Total	*1.555*	*100%*	*1,276.9*	*100%*	*0.351*	*100%*	*68.8*	*100%*

Table 2. Comparison of GPU-enabled routines and single-threaded routines for sample road.

separate threads analyse specific grid cells simultaneously. Furthermore, our improvements using GPUs are a preliminary improvement. It is anticipated that through more carefully-considered use of shared memory and the optimal arrangement of threads into thread blocks on newer model GPUs we may be able to achieve higher performance gains.

4 CONCLUSIONS

This paper presented formulations of the three most computationally-intensive components of optimal road design through ecologically sensitive regions. Due to their highly parallel nature, these components are well-suited to being implemented on GPUs, as evidenced by the dramatic improvements in speed over CPU-implementations of the same problem. Together with the use of surrogate models presented in other research, GPUs will allow road designers to dramatically reduce the time required to search for high-value roads through ecologically-sensitive regions while optimally controlling traffic flow over time. It is anticipated that the GPU implementation of the optimal control component with endogenous uncertainty will also be useful to researchers in the areas of real options and optimal control more broadly.

ACKNOWLEDGEMENT

This research was funded through an Australian Government Research Training Program Scholarship and Data61 top-up scholarship.

REFERENCES

Davey, N., S. Dunstall, and S. Halgamuge (2016). The value of flexible road designs through ecologically sensitive areas. In *Lecture Notes in Management and Industrial Engineering*, Canberra, Australia. In press.

Davey, N., S. Dunstall, and S. Halgamuge (2017). Optimal road design through ecologically sensitive areas considering animal migration dynamics. *Transp. Res. Part C Emerg. Technol. 77*, 478–494.

Davey, N., N. Langrene, S. Dunstall, J. Rhodes, S. Halgamuge, and C. Davey (2018). The conservation impacts of optimal road design with operating flexibility.

Jha, M. K., P. Schonfeld, and J. C. Jong (2006). *Intelligent Road Design*, Volume 19. WIT Press.

Jong, J. C. and P. Schonfeld (2003). An evolutionary model for simutaneously optimizing three-dimensional highway alignments. *Transportation Research Part B 37*(2), 107–128.

Kang, M. W., M. K. Jha, and P. Schonfeld (2012). Applicability of highway alignment optimization models. *Transportation Research Part C 21*, 257–286.

Kharroubi, I., N. Langrené, and H. Pham (2014). A numerical algorithm for fully nonlinear HJB equations: an approach by control randomization. *Monte Carlo Methods Appl. 20*(2), 145–165.

Langrené, N., T. Tarnopolskaya, W. Chen, Z. Zhu, and M. Cooksey (2015). New regression Monte Carlo methods for high-dimensional real options problems in minerals industry. In *MODSIM2015, 21st International Congress on Modelling and Simulation*, Gold Coast, Australia, pp. 1077–1083.

Lee, A., C. Yau, M. B. Giles, A. Doucet, and C. C. Holmes (2010). On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *Journal of computational and graphical statistics 19*(4), 769–789.

Longstaff, F. A. and E. S. Schwartz (2001). Valuing American options by simulation: A simple least-squares approach. *The Review for Financial Studies 14*(1), 113–147.

Nickolls, J., I. Buck, M. Garland, and K. Skadron (2008). Scalable parallel programming with CUDA. *Queue 6*(2), 40–53.

Ryoo, S., C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu (2008). Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 73–82. ACM.

Sampathirao, A. K., P. Sopasakis, A. Bemporad, and P. Patrinos (2015). Distributed solution of stochastic optimal control problems on gpus. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pp. 7183–7188. IEEE.