Full Implementation of an Estimation of Distribution Algorithm on a GPU

CIGPU GPU Competition Entry

Simon Poulding
Department of Computer Science
University of York
Heslington, York, UK
smp@cs.york.ac.uk

Jan Staunton
Department of Computer Science
University of York
Heslington, York, UK
jps@cs.york.ac.uk

Nathan Burles
Department of Computer Science
University of York
Heslington, York, UK
nburles@cs.york.ac.uk

1. INTRODUCTION

We submit an implementation of an Estimation of Distribution Algorithm – specifically a variant of the Bayesian Optimisation Algorithm (BOA) – using GPGPU. Every aspect of the algorithm is executed on the device, and it makes effective of use multiple GPU devices in a single machine. As for other EDAs, our implementation is generic in that it may be applied to any problem for which solutions may be represented as binary strings. For the purpose of this paper, we apply it to a particular problem known to be difficult for metaheuristic algorithms due to high interdependency between variables: finding the lowest energy state of an Ising Spin Glass. We show that our GPU implementation demonstrates a speedup in excess of 80x compared with an equivalent CPU implementation. To our knowledge, this is the first EDA to be implemented fully on the GPU.

2. BACKGROUND AND PREVIOUS WORK

2.1 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) are population-based probabilistic search techniques that search solution spaces by learning and sampling probabilistic models [4]. EDAs iterate over successive populations of candidate solutions to a problem. Each population is sometimes referred to as a generation. To construct a successor population, EDAs build a probabilistic model of promising solutions from the current population and then sample that model to generate new individuals. The newly generated individuals replace individuals in the current population to create the new population.

Algorithm 1 Pseudocode for basic EDA

```
P = InitialPopulation();
while not(termination_criterion) do
    evaluate(P);
    S = SelectPromisingSolutions(P);
    M = UpdateModelUsingSolutions(S);
    N = SampleFromModel(M);
    P = ReplaceIndividuals(N);
end while
```

The pseudocode of a basic EDA algorithm is shown in Algorithm 1. Readers who are familiar with Genetic Algorithm (GA) literature can view EDAs as similar to a GA with the crossover and mutation operators replaced with the model building and sampling steps. EDAs can be seen as strategically sampling the solution space in an attempt to find a "good" solution whilst learning a model of "good" solutions along the way. EDAs are sometimes referred to as Probabilistic Model-Building Genetic Algorithms (PMBGAs), a full overview of which can be found in [4].

For this entry, we have implemented a variant of the Bayesian Optimisation Algorithm (BOA) [3] on the GPU. BOA uses Bayesian networks to model the relationship between values in binary string solutions. In order to construct the Bayesian network given a set of desirable binary strings, a greedy network construction algorithm is used that adds edges between parents and child nodes (where each node represents a single binary value). Our implementation makes use of K2 metric [1] to determine the next edge to add in order to derive a model that best represents the "good" solutions in the current population.

Restricted Tournament Replacement (RTR) is employed as part of our implementation to encourage diversity in the population during the course of any given search. To replace a solution with a new solution i in a population P during the replacement phase of our EDA, RTR chooses a random subset S from P, then finds the "most similar" solution j to i in S, where similarity is determined by the Hamming distance of the binary string representations. If i is better than j, then replace j with i in P.

2.2 Parallelisation of EDAs

Existing work that parallelises EDA model building across CPUs makes use of an ancestral node ordering to allow the model to be built independently for each node in the Bayesian network [2]. The ordering is generated randomly at each iteration and specifies which nodes can be parents of a given child node, and thus avoid cycles in the network when the model is constructed in a distributed manner. We make use of this ancestral node ordering in our implementation.

2.3 Ising Spin Glasses

Our chosen example problem are Ising Spin Glasses (ISG), which are often used as benchmark problems for sophisticated EDAs such as BOA [5]. ISGs are a mathematical model used in statistical physics. A number of particles exist in a spin-up ($S_i = +1$) or spin-down ($S_i = -1$) state. The particles are typically arranged in a lattice: our implementation considers toroidal 2D and 3D lattices. Each particle is coupled with its immediate neighbours in the lattice according to coupling constants, J, which also take values of -1 or $+1$. The free energy, E, of the particles is derived using the equation:

$$E = -\sum_{i,j} J_{ij} S_i S_j \tag{1}$$

where sum occurs over all neighbouring particles S_i and S_j.

The objective is to find the lowest energy configuration of particle spins given a set of coupling constants (which in our case are randomly assigned). This problem is particularly difficult for many optimisation techniques owing to the high interdependency of particle spins, and thus the non-decomposability of the problem.

3. IMPLEMENTATION

The EDA algorithm is implemented entirely on the GPU device under the control of a CPU process. The chosen fitness function for our chosen example of Ising Spin Glasses is also implemented on the GPU, with all processing occurring on the GPU under the control of a CPU process. The algorithm is implemented using the CUDA 4.0 SDK, and makes use of the Thrust library for standard operations such as sorting.

The model building phase of the algorithm is by far most computationally intensive step for large problem sizes. It is this step that can be distributed to multiple GPUs, allowing for near linear speed up over a single GPU card implementation. The work is distributed proportionally according to the computational power of the GPGPU capable cards, so as to avoid a slow card limiting overall performance.

This step takes advantage of the ancestral node ordering described in section 2.2 to enable the model to be built independently for each node in the Bayesian network. In the GPU implementation, each CUDA block is assigned to the model building for one node, resulting in significant parallelisation of this step and thereby effective utilisation of the GPU.

As well as scaling to use multiple GPU devices, we also believe the implementation scales to different cards architectures since parameters, such as grid and block topologies, are derived based on queried device properties.

4. EVALUATION

In order to show the scaling properties of the GPU implementation, we compare to an equivalent CPU implementation. The problem used is the 3D Ising Spin Glass problem described in section 2.3. We performed experiments on two parameters, the problem size and the population size. The results are shown in the tables below, and are calculated from an average of 10 trials (runs) for each parameter combination. The CPU implementation used 4 threads, whilst the GPU implementation used 4 Tesla cards. Any results with a * next to them are the result of one experiment since it was impractical to run these lengthy trials multiple times. The pertinent parameters are as follows unless specified in the tables: Spin Glass Size: 16x16x16, Population Size: 1024, Maximum Parents: 4, Iterations: 20, Best Population Size: $\frac{1}{8}$ Population, Number Replaced: $\frac{1}{4}$ Population, RTR Tournament Size: 16.

Table 1: Problem Size Results

Problem Size Results				
Problem Size	8	12	16	24
GPU	0.3s	1.08s	4.45s	41.14s
CPU	2.37s	23.46s	217.22s*	3391.25s*
Speed up	7.9x	21.72x	48.81x	81.82x

Table 2: Population Size Results

Population Size Results				
Max Parents	512	1024	2048	4096
GPU	2.75s	4.55s	4.98s	9.98s
CPU	77.16s	208.56s	307.64s	-
Speed up	28.05x	45.84x	61.78x	-

As can be seen, the speedup provided by GPU implementation increases from 8x to more than 80x as the problem size increases to spin glasses of size 24x24x24. Similarly, the GPU speedup increases with population size.

5. CONCLUSION

To conclude, we have implemented an EDA fully on the GPU. We have shown that the GPU implementation can be up to 80 times faster in some cases, and scales well with respect to the problem size and population size. The implementation can use multiple cards on a single host and uses preprocessing in order to make the GPU implementation more efficient. To our knowledge, this is the first sophisticated BOA-like EDA to be fully-implemented on the GPU. Please find more details in the help files of the implementation package submitted with this report. Our future aims are to use this implementation in order to perform research into larger practical problems.

6. REFERENCES

[1] C. Borgelt and R. Kruse. An empirical investigation of the K2 metric. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 2143 of *Lecture Notes in Computer Science*, pages 240–251. 2001.

[2] J. Očenášek and J. Schwarz. The distributed bayesian optimization algorithm for combinatorial optimization. In *Proc. EUROGEN 2001, Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, pages 115–120, 2001.

[3] M. Pelikan, D. Goldberg, and E. Cantu-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume 1, pages 525–532, 1999.

[4] M. Pelikan, D. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational optimization and applications*, 21(1):5–20, 2002.

[5] M. Pelikan, M. Pelikan, D. E. Goldberg, and D. E. Goldberg. Hierarchical boa solves ising spin glasses and maxsat. In *In Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2003), number 2724 in LNCS*, pages 1271–1282. Springer, 2003.