Efficient Design Space Exploration of GPGPU Architectures

Ali Jooya, Amirali Baniasadi, and Nikitas J. Dimopoulos

Department of Electrical and Computer Engineering
University of Victoria
Victoria, B.C., Canada
{jooya,amirali,nikitas}@ece.uvic.ca

Abstract. The goal of this work is to revisit GPU design and introduce a fast, low-cost and effective approach to optimize resource allocation in future GPUs. We have achieved this goal by using the Plackett-Burman methodology to explore the design space efficiently. We further formulate the design exploration problem as that of a constraint optimization. Our approach produces the optimum configuration in 84% of the cases, and in case that it does not, it produces the second optimal case with a performance penalty of less than 3.5%. Also, our method reduces the number of explorations one needs to perform by as much as 78%.

1 Introduction

Employing Graphics Processing Units (GPUs) as accelerators for general-purpose throughput-intensive workloads has become an important part of high performance computing. GPUs' [1] computing power stems from hundreds of processing elements referred to as PEs. A group of several PEs comprise a Stream Multiprocessor (SM). Multiple SMs are grouped in scalable arrays and connected to memory controllers through an interconnection network. In addition to the PEs, an SM includes three separate L1 caches referred to as constant cache, texture cache and data cache. Other components forming an SM are shared memory, register file and thread pool. Figure 1 shows a typical GPU.

As technology advancements provide designers with a growing chip real-estate, effective methods of exploiting the available resources becomes essential.

The goal of this work is to revisit GPU design and introduce a fast, low-cost and effective approach to optimize resource allocation in future GPUs. Our solution utilizes the Plackett-Burman methodology [2] and formulates a constraint optimization problem to find the optimal GPU design for different available chip real-estate budgets without resorting to exhaustive design space exploration. Our results show that for the applications and configurations investigated, our solution matches the design suggested by an exhaustive search 48 out of 57 times, and for the cases the proposed solution did not match the one found by the exhaustive search, the error is less than 3.5%.

The remaining of this paper is organized as follows. In Section 2 we review the background techniques used. In Section 3 we discuss our methodology. In Section 4 we present the results and we conclude with Section 6.

I. Caragiannis et al. (Eds.): Euro-Par 2012 Workshops, LNCS 7640, pp. 518–527, 2013.

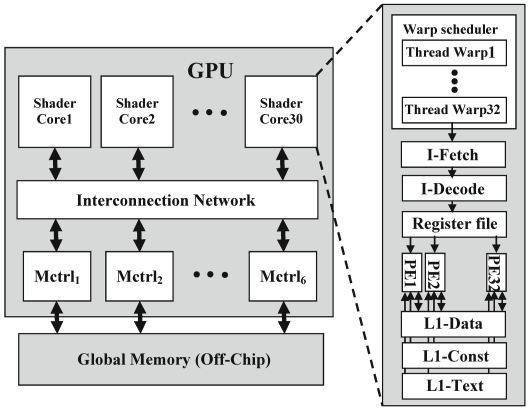

Fig. 1. The structure of a GPU

2 Background

2.1 Plackett-Burman Design Methodology

In design space exploration, one needs to conduct numerous experiments varying a number of parameters to ensure full exploration. In designs with N parameters, each one taking one of L values, L^N experiments need to be conducted. Plackett and Burman [2] introduced a method which can measure the effects of the N parameters, where multi-parameter interaction is not present, by conducting X experiments, where X is the next multiple of 4 strictly greater than N. A Plackett-Burman design with fold-over, can quantify the effect of two interacting parameters. It has been claimed [3] that in the case of processor design exploration with less than 10 parameters, single and two-parameter interactions have the most significant effect, and since we use only four parameters in this work, we shall utilize a Plackett-Burman design with fold-over.

Plackett and Burman [2] introduced templates of experiments for different number of parameters. Table 1 shows an example of such a template (with fold-over). The rows correspond to the experiments while the columns correspond to the parameters. A "+" ("−") sign indicates that for the experiment indicated, the corresponding parameter is set to its maximum (minimum) value.

For each experiment i, a target performance metric T_i is measured. After running all experiments, it is possible to calculate the effect of each parameter on the design. We denote by S_α the effect of parameter α on the target performance metric. S_α can be calculated by subtracting the target values corresponding to the low parameters from those when they are high. For example, from Table 1, the effect S_E of parameter E is calculated as:

$$S_E = T_1 - T_2 + T_3 + T_4 + T_5 - T_6 - T_7 - T_8 - T_9 + T_{10} - T_{11} - T_{12} - T_{13} + T_{14} + T_{15} + T_{16}$$

The magnitude of the effect is used to identify the parameter that is most important in determining performance.

Table 1. Placket-Burman design with fold-over; 7 parameters, 16 experiments

experiment	parameters							Target	experiment	parameters							Target
	A	B	C	D	E	F	G	performance		A	B	C	D	E	F	G	performance
1	+	+	+	-	+	-	-	T_1	9	-	-	-	+	-	+	+	T_9
2	-	+	+	+	-	+	-	T_2	10	+	-	-	-	+	-	+	T_{10}
3	-	-	+	+	+	-	+	T_3	11	+	+	-	-	-	+	-	T_{11}
4	+	-	-	+	+	+	-	T_4	12	-	+	+	-	-	-	+	T_{12}
5	-	+	-	-	+	+	+	T_5	13	+	-	+	+	-	-	-	T_{13}
6	+	-	+	-	-	+	+	T_6	14	-	+	-	+	+	-	-	T_{14}
7	+	+	-	+	-	-	+	T_7	15	-	-	+	-	+	+	-	T_{15}
8	-	-	-	-	-	-	-	T_8	16	+	+	+	+	+	+	+	T_{16}
										S_A	S_B	S_C	S_D	S_E	S_F	S_G	

2.2 Knapsack Problem

The knapsack problem is a constraint optimization problem [4]. Denoting by v_j (w_j) the value (weight) of an item of *typej*, b_j the maximum number of items of *typej* and C the capacity of the knapsack, the knapsack problem is stated as:

Select x_j *(j = 1, . . . , n)* items of *typej* so as to maximize $z = \sum_{i=1}^{n} v_j x_j$ subject to $\sum_{j=1}^{n} w_j x_j \leq C$; $1 \leq x_j \leq b_j$ and integer; $j \in N = 1, \ldots, n$.

This is a Bounded Knapsack Problem (BKP) [4], a generalization of the 0-1 knapsack problem. We assume that v_j, w_j, b_j and C are positive integers.

3 Methodology

Our objective is to find the GPU configuration that achieves maximum performance under a given transistor budget. Such problem can be solved by an exhaustive designn space exploration, which is computationally expensive.

Our approach formulates the problem as a knapsack problem which can be solved efficiently [5]. In formulating the knapsack problem, one needs to identify the *value* (v_j) and *weight* (w_j) parameters. While the *weight* parameter is easily identifiable as the transistor count of each unit, the *value* parameter is not as directly identifiable. The *value* parameter simply shows the contribution of each of the architectural units to the performance of the overall architecture and for a particular application. This information is not readily available. We postulated that within the domain of exploration, the contribution of each unit to the performance is proportional to its cardinality and or size. The proportionality coefficient (i.e. the *value*) though is unknown. In this work, we shall show that the effect S_α of parameter α as per Plackett-Burman, can be used as the *value*.

3.1 Establishing the Transistor Count

Configuring a GPU involves choosing the number of memory controllers, the size of the data cache, the size of the register file and the size of the constant cache among other parameters.

We estimated the cost (in number of transistors) for the data cache, the constant cache and the register file following the method cited in [6], while for the cost of the memory controller, we used the cost of a Xilinx memory controller core [7]. The cost for each unit are shown in Table 4.

3.2 Benchmarks

We used the following set of GPGPU benchmarks from NVIDIA's CUDA software development kit (SDK) [8] and Rodinia [9] benchmark suits: AES Cryptography (AES); Fast Walsh Transform (FWT); LIBOR Monte Carlo (LIB); 3D Laplace Solver (LPS); Montcarlo; Neural Network Digit Recognition (NN); Scan; Srad; Black schole; Hotspot; Ray tracing (Ray); Matrix multiplication (Matrix); Back propagation (Backprop).

These benchmarks were compiled and run on the GPGPU-Sim [10] simulator with the system configuration shown in Table 2. The ranges of the sizes or cardinality of the units used in configuring the GPU for our experiments, are shown in Table 3. The variation in the values of these parameters resulted in a large number of possible configurations. Totally, we experimented with 553 configurations. The simulations result in performance estimates for each configuration and for each benchmark, reported as *instructions per cycle* (IPC).

3.3 Use of Plackett-Burman and Optimization

In order to determine the *value* parameters, we applied the Plackett-Burman methodology with fold-over for each benchmark. The number of parameters used was four (i.e. number of memory controllers, the size of the data cache, the size of the register file and the size of the constant cache).

The resulting effect values S_α are used as the *value* parameters for the knapsack optimization problem while as the *weight* parameters we used the transistor counts for each of the units as discussed in Section 3.1 above.

We postulated earlier that the contribution of each of the units relates linearly to the performance obtained. Figure 2 shows the performance (as IPC) as a function of the number of memory controllers and for different sizes of the DL1 cache for the LIB benchmark. The linear dependence of the performance on the number of controllers is evident for most of the domain of exploration.

We used MATLAB's[1] linprog function to solve the knapsack problems.

4 Evaluation and Results

As we discussed in Section 3.3 we used four parameters: memory controller block, DL1 cache, constant cache and register file. Table 4 shows the estimated costs of the parameters in terms of transistor counts while Table 3 shows the ranges of the

[1] MATLAB is a registered trademark of The Math Works, Inc. linprog is in the optimizetion toolbox.

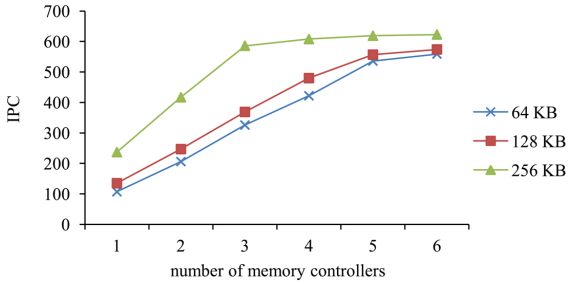

Fig. 2. IPC of LIB benchmark for different sizes of DL1 cache and number of memory controller blocks

Table 2. GPU configuration

Parameter	Value	Parameter	Value
Number of shaders	30	Warp width	32
Shader clock frequency	1.3 GHZ	Scheduling	PDOM
Max thread per shader	1024	max CTA/Shader	8
SIMD pipline width	32		

configuration parameters. Note that not all benchmarks have constant memory access instructions. Therefore, we have evaluated our method using three or four parameters depending on the benchmark under simulation. We have limited the number of memory controller blocks (MCB) to six, as more than five MCBs do not affect performance for most benchmarks.

We run the optimization problem for a gradually increasing transistor budget and for each budget, we obtain the resulting optimum-performance configuration. Figure 3 shows the resulting configurations for the NN benchmark. The horizontal axis refers to the transistor budget while the vertical axis shows the number of units corresponding to the optimum configuration. In the figure, one can distinguish regions of the transistor budget where the configuration does not change. For the NN benchmark (please refer to Figure 3) we distinguish five regions marked by numbers 1 to 5 on the figure.

As an example, for a transistor budget of no more than 67 million transistors (i.e. within region 2), the optimum configuration comprises four memory controller blocks (MCBs), 2 units of DL1, one unit of Constant Cache and one unit of Register file. Some explanation is necessary here as to the definition of a unit. While the memory controller blocks are self evident, the units of the other parameters vary with the benchmark. For example for the NN benchmark (please refer to Table 3), a unit of DL1 Cache is 8KB, a unit of Constant Cache is 512B while a unit of Register File is 4KB. In this work, we have used as a unit the minimum size considered for each configuration parameter and for each benchmark, and then allowed the optimization function to return as a solution a configuration comprising multiples of the units of each parameter. For the said

example, the optimum configuration obtained comprises 4 Memory Controller Blocks, 16 KB of DL1 Cache, 512B of Constant Cache, and a 4KB register file.

Please note that the caches and register files can only be configured in sizes that are power of two. If the solution of the optimization returned a configuration that had a parameter that violated the "power of two rule", then the size of this parameter was fixed to a power of two that was less than the size suggested by the optimization, the excess transistor budget was returned to the transistor budget, and another optimization was run on the remaining parameters.

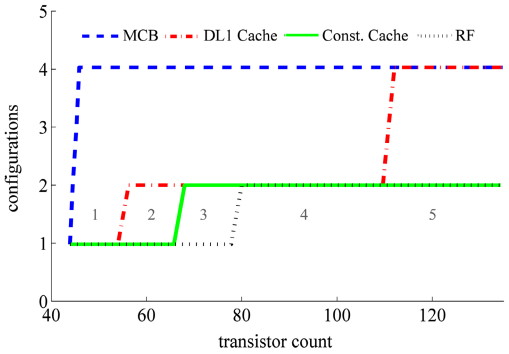

Fig. 3. Optimum configuration regions for the NN benchmark

To ensure that the configurations suggested by the optimization method are indeed optimal, we also performed an exhaustive search of all possible configurations adhering to the transistor budgets considered and compared the results. We shall analyze these results in Section 4.1 below. To conclude the example of the NN benchmark, Table 6 presents all possible configurations that adhere to the transistor budget of 67 million. As it can be verified, the configuration suggested by the optimization solution (Conf12) has indeed the optimum performance. Please note that because of space limitations, this table does not include configurations in Region 1 (i.e. with transistor budgets of less than 56 million). All configurations in region 1 have a lesser performance as compared to Conf12.

4.1 Results

Table 5 shows the Placket-Burman results for the benchmarks. Each column marked with the name of the parameter reports the effect S_α of the said parameter. The columns marked by "Rank" represent the rank of the effect of the parameter to its left. Thus, for the NN benchmark, we see that the DL1 Cache is ranked as first, that is this benchmark's performance is influenced primarily by the size of its DL1 Cache. It is interesting to note that for most of the benchmarks, the number of memory controllers influences the performance the most,

Table 3. Benchmarks and low and high values of resources

Benchmark	MCB	DL1 Cache	Const Cache	Register File
AES	1-3	1KB,2KB	512B-8KB	4KB,8KB
FWT	1-4	4KB-32KB	N/A	4KB,8KB
LIB	3-5	32KB-128KB	64KB-256KB	2KB,4KB
LPS	1-3	1KB,2KB	N/A	2KB,4KB
Montcarlo	1-5	32KB,64KB	1KB,2KB	8KB,16KB
NN	1-4	8KB-32KB	512B,1KB	4KB,8KB
Scan	1-3	512B-2KB	N/A	2KB-8KB
Srad	1-6	2KB-256KB	N/A	4KB-16KB
Blackschole	1-4	2K-8K	N/A	4K,8K
Hotspot	1,2	1K,2K	N/A	8K,16K
Ray	1-3	1K,2k	1K-32K	8K,16k
Matrix	1-5	1K,2k	N/A	4K,8K
Backprop	1-3	1K,2k	N/A	4K,8K

Table 4. Transistor cost (in millions of transistors) of resource units

Parameter	Size/Number	Cost	Parameter	Size/Number	Cost
memory controller	1	0.3	Constant cache	32KB	52
DL1 cache	32KB	87	Register file	32KB	170

indicating that the higher memory bandwidth may improve performance significantly, seconded by the size of the DL1 cache, while the sizes of the Constant Cache and the Register File have a lesser impact. However, for some benchmarks, e.g., NN, Ray, LIB and AES, the parameter's rank follows a different order. For example, AES is composed of a single kernel which has 257 blocks of 8 warps. Two blocks can run concurrently on each shader which increases the significance of the number of memory controllers to provide sufficient bandwidth. As AES is optimized to store constants in constant memory the performance is sensitive to the size of constant memory. On the other hand, few local memory accesses reduce the sensitivity of performance to the DL1 cache. LIB runs two kernels, each having 64 blocks of 64 threads. Excessive register file usage of each thread limits the number of concurrent blocks. Similar explanations on the parameter sensitivity can be obtained for the other benchmarks.

Figure 4 shows the performance (as IPC) of the best (grey bars) and ILP-suggested (black bars) configurations for all regions of each benchmark. As it can be seen, the optimization algorithm suggested solutions match the optimum performance ones as found through exhaustive search of all possible configurations, except for nine cases. Namely the solutions suggested by the optimization procedure for Regions 1 and 4 of the AES and Regions 4 and 5 of the Srad and Regions 2, 3, 4, 6 and 8 of the Ray benchmarks, have performance that is slightly lower than the optimal solution found through exhaustive search. For these cases, the performance of the configuration obtained by the optimization

Table 5. Resulting parameter effects after Plackett-Burman for all benchmarks

Bench.	MCB	Rank	DL1 cache	Rank	const cache	Rank	RF	Rank
AES	57100	1	17742	4	35500	2	18922	3
FWT	127799	1	73903	2	—	-	459	3
LIB	1482938	3	2284182	2	1051960	4	6037368	1
LPS	340704	1	136082	2	—	-	16736	3
Montcarlo	445073	1	293537	2	46469	3	26229	4
NN	1054553	2	1508787	1	5307	3	535	4
Scan	12372	1	9708	2	—	-	1748	3
Srad	139159	1	2125	2	—	-	117	3
Blackschole	3110257	1	142613	2	—	-	29369	3
Hotspot	433926	1	133392	2	—	-	4584	3
Ray	34441615	2	38417897	1	1405	4	5684461	3
Matrix	13693	1	7931	2	—	-	1775	3
Backprop	14864	1	12020	2	—	-	476	3

method is very close to the optimum (less than 3.5% difference in performance). Also, in all these cases, the ILP-based solution had the second best performance.

5 Related Work

Wilson et al. [11] proposed a technique to deal with divergent branches by forming new warps dynamically using threads that take different execution paths at the divergence, increasing the system throughput.

Jia et al. [12] proposed an automated GPU performance exploration framework, named Stargazer, based on stepwise regression modelling. Stargazer simulates design points which have been sampled randomly from a full GPU design

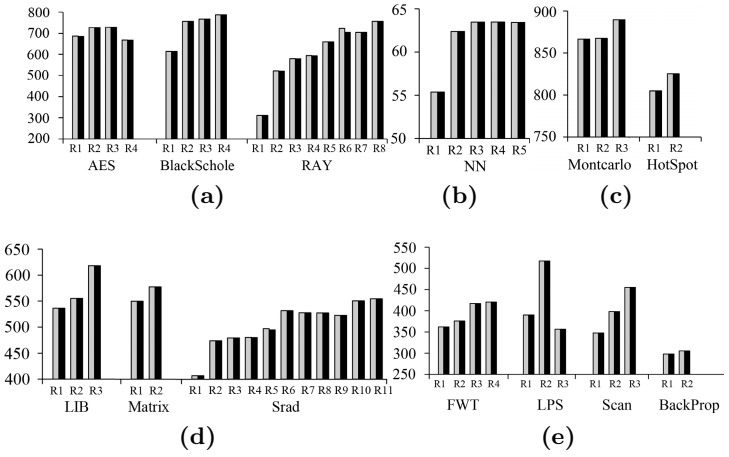

Fig. 4. IPC Comparison between the ILP-suggested and the best performing configurations for different benchmarks

Table 6. All configurations for region 2 of the NN benchmark

parameter	Conf1	Conf2	Conf3	Conf4	Conf5	Conf6
memory controller	1	1	1	1	2	2
constant cache	512B	512B	1KB	1KB	512B	512B
DL1 cache	8KB	16KB	8KB	16KB	8KB	16KB
register file	8KB	4KB	8KB	4KB	8KB	4KB
IPC	39.32	52.71	39.74	52.59	48.78	59.28
cost	65.36	65.86	66.17	66.67	65.66	66.16
parameter	Conf7	Conf8	Conf9	Conf10	Conf11	Conf12
memory controller	2	3	3	3	4	4
constant cache	1KB	512B	512B	1KB	512B	512B
DL1 cache	8KB	8KB	16KB	8KB	8KB	16KB
register file	8KB	8KB	4KB	8KB	8KB	4KB
IPC	48.90	52.83	61.55	52.49	54.98	**62.38**
cost	66.47	65.96	66.46	66.77	66.26	66.76

space and uses these sample simulation to build a performance estimator. The results showed that using 0.03% of full design space, Stargazer predicts the performance of any design point with less than 1.1% error.

Palermo et al. [13] proposed a DSE methodology for application-specific multiprocessor system-on-chip. The proposed methodology uses design of experiment techniques, e.g. random and full factorial, to find a set of good candidate architecture configurations to minimize the number of simulations. Then using response surface modeling and the simulation results, an analytical representation of the system's objective function is generated.

Couvreur et al. [14] applied MMKP to design a MP-SoC runtime manager. Whenever the environment is changing, the runtime manager considers the specification of available application mappings, information provided by the design time exploration, the platform information, and the user requirements to select operating points that minimize total energy consumption.

Lilja et al. [3] applied Plackett and Burman design to identify key processor parameters and analyzed their effect on processor enhancement.

6 Conclusions

In this work, we presented a method that establishes a GPU configuration that adheres to a transistor budget limit and obtains the optimum or near optimum performance. Among all parameters in GPU design space, we used four parameters, i.e. register file size, shared memory and data cache sizes and number of memory controller, that we could estimate the costs in term of transistor count. We did not include the remaining of the parameters, because of the difficulty in calculating the costs.

Our method delivered the optimum performing configuration in 48 out of 57 cases, and for the nine cases where the configuration did not achieve optimum

performance, its performance lagged the optimum one by less than 3.5%. Our method is very efficient in that it requires far fewer simulations to achieve its results. For example, for the case of the Srad benchmark, an exhaustive design exploration would have required the simulation of 144 configurations, while our method arrives at the optimum configuration by simulating just 16.

Further, we plan to investigate the applicability of the method discussed in this work to designs where the constraint is power rather than transistor count.

Acknowledgment. Computational support was provided through Compute/ Calcul Canada. This work was supported in part by NSERC, the Lansdowne Chair in Computer Engineering at UVic, and IBM through a Faculty Award.

References

1. Nickolls, J., Dally, W.J.: The GPU Computing Era. IEEE Micro 30(2), 56–69 (2010)
2. Plackett, R., Burman, J.: Design of optimum multifactorial experiments. Biometrika 33(4), 305–325 (1944)
3. Yi, J.J., Lilja, D.J., Hawkins, D.M.: Improving computer architecture simulation methodology by adding statistical rigor. IEEE Transactions on Computers 54(11), 1360–1373 (2005)
4. Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons (1990)
5. Parra-Hernandez, R., Dimopoulos, N.J.: A new heuristic for solving the multichoice multidimensional knapsack problem (MMKP). IEEE Trans. on Systems Man and Cybernetics Part A 35(5), 708–717 (2005)
6. Steinhaus, M., Kolla, R., Larriba, J., Ungerer, T., Valero, M.: Transistor count and chip-space estimation of simplescalar based microprocessor models. In: Workshop on Complexity-Effective Design (2001)
7. Spartan: LogiCORE IP multi-port memory controller (v6.04.a) (July 6, 2011)
8. NVIDIA: CUDA SDK code samples (v4)
9. Che, S., Boyer, M., Meng, J., Tarjan, D., Lee, S., Sheaffer, J.W., Skadron, K.: A benchmark suite for heterogeneous computing. In: IEEE International Symposium on Workload Characterization (October 2009)
10. Bakhoda, A., Yuan, G., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing CUDA workloads using a detailed GPU simulator. In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (April 2009)
11. Fung, W., Sham, I., Yuan, G., Aamodt, T.: Dynamic warp formation and scheduling for efficient GPU control flow. In: 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2007, pp. 407–420 (December 2007)
12. Jia, W., Shaw, K.A., Martonosi, M.: Stargazer: Automated regression-based GPU design space exploration. In: IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2012 (2012)
13. Palermo, G., Silvano, C., Zaccaria, V.: Respir: A response surface-based pareto iterative refinement for application-specific design space exploration. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28(12), 1816–1829 (2009)
14. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In: International Symposium on System-on-Chip 2006, pp. 1–4 (November 2006)