CUDA tuning and configuration parameters on Fermi architecture

Yuri Torres, Arturo Gonzalez-Escribano and Diego R. Llanos

Universidad de Valladolid, Spain

`{yuri.torres|arturo|diego}@infor.uva.es`

INTRODUCTION

- **Modern graphics processing units** used as general purpose architectures.

- **NVIDIA CUDA architecture and model** [1]. Objective: Simplify the encoding of parallel, general-purpose applications, on heterogeneous systems with GPUs devices.

- **Fermi**: The NVIDIA's latest generation of CUDA architecture [2]. Presents a *improved double precision performance*; a *transparent cache hierarchy*; *variable-size shared memory*; and *faster atomic operations*.

- **CUDA programming model** forces to select the *threadblock size* and *shape*.

- **Occupancy**: Ratio between active warps and maximum number of warps in a SM. Determined by threadblock.

- **Our goal:** relationship among global parameters configuration (*threadblock size and shape*), *Occupancy* and *threads access pattern* in global device memory [3].
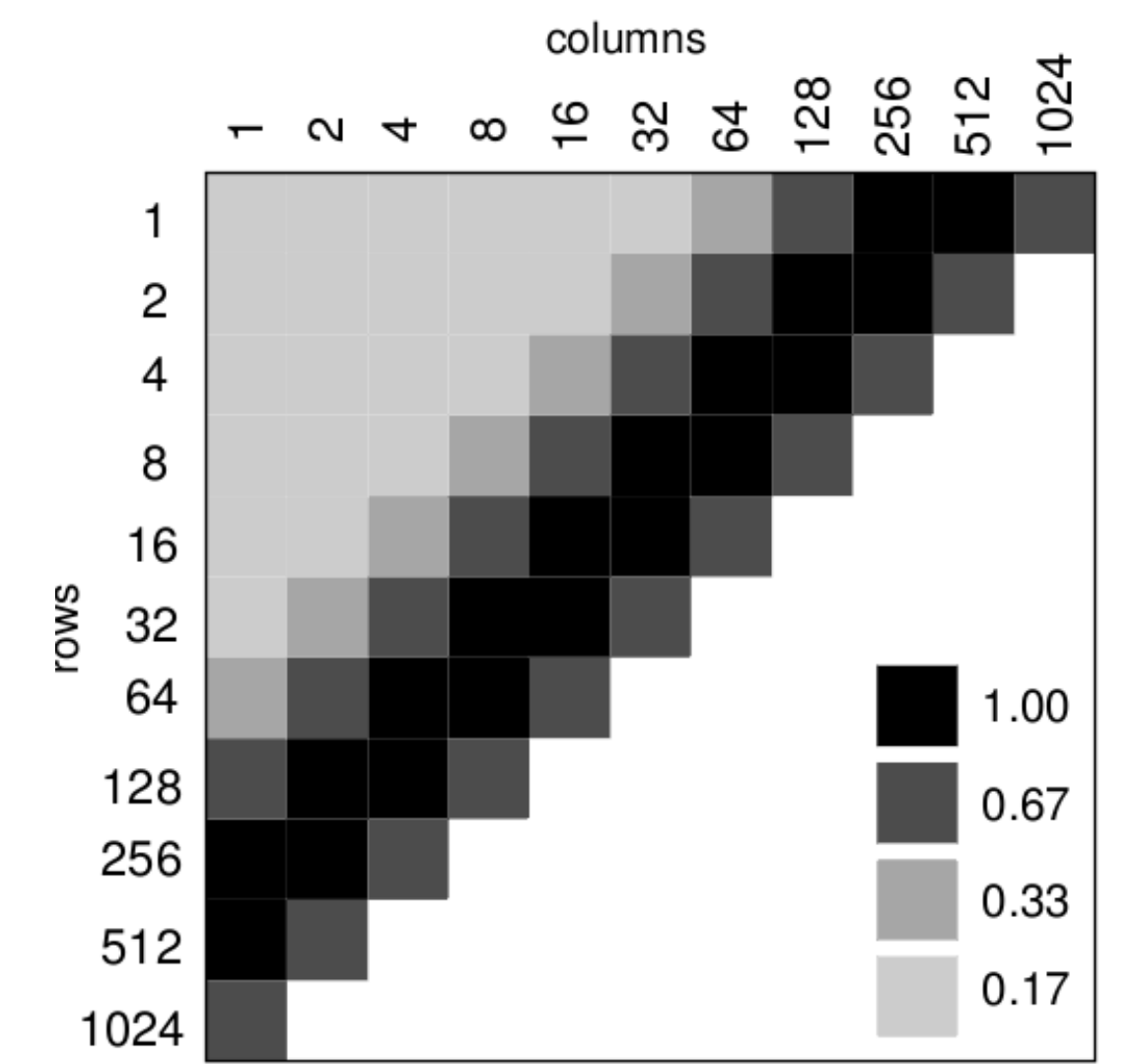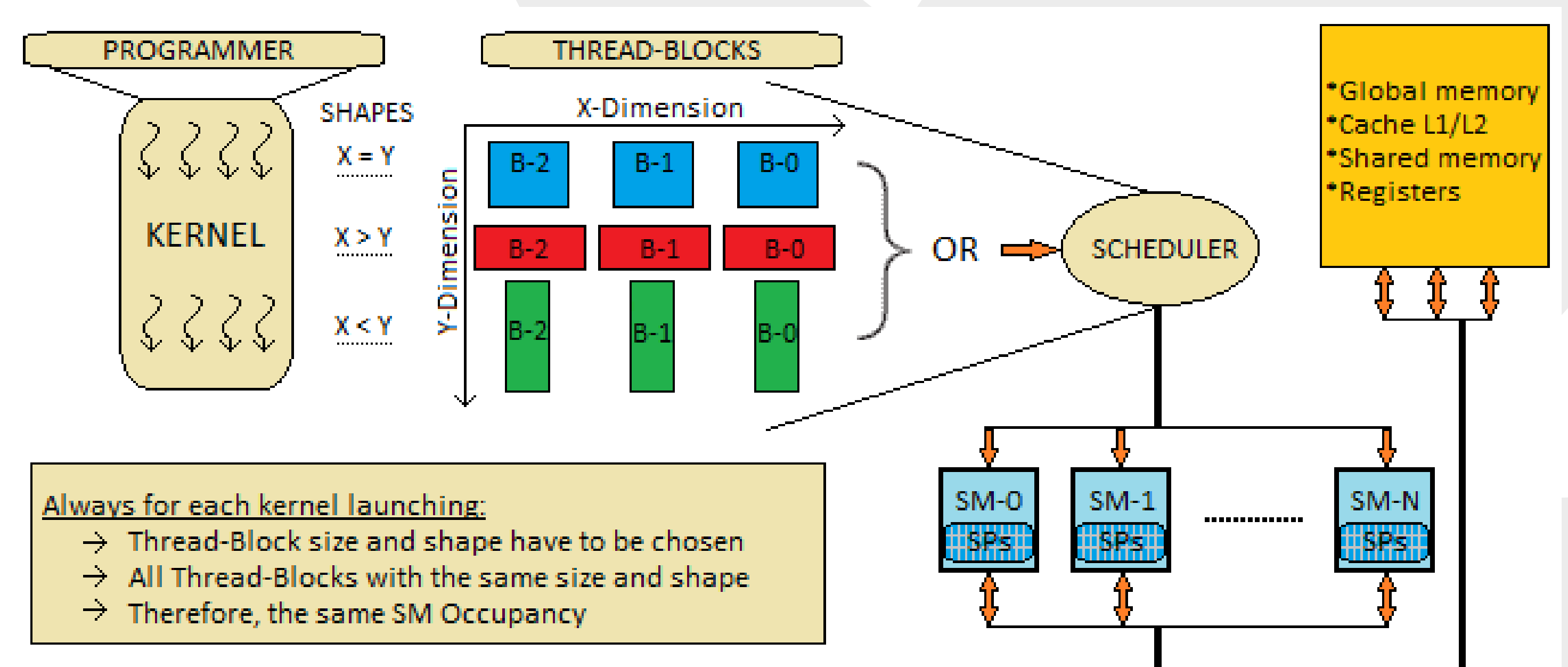

Fig: Occupancy of different threadblock

FERMI ARCHITECTURE

Parameter	pre-Fermi	Fermi
SPs (per-SM)	8	32
Registers (per-SM)	16 KB	32 KB
Max. number of threads (per-SM)	1024	1536
Max. number of threads (per-block)	512	1024
Warp size	32	32
Warp scheduler	Single	Dual
Shared memory banks (per-SM)	16	32
L1 cache (per-SM)	-	0/16/48 KB
L2 cache	-	764 KB
Global memory banks	8	6
Size of global memory transaction	32/64/128 B	32/128 B

THREADBLOCK SIZE AND SHAPE

EXPERIMENTS

- **Matrix addition**: a bi-dimensional problem ($C = A + B$). Each threads present a easy memory access pattern accessing to contiguous global memory positions (coalesced and no-data reutilization implementation).

- **Naive matrix multiplication**: a bi-dimensional problem ($C = A \times B$). Each thread is associated with a single C position. The internal global memory accesses are inefficient and more complicated (no full coalesced and data reutilization implementation).

RESULTS

	1	2	4	8	16	32	64	128	256	512	1 024
1	230.66	115.64	58.13	30.1	15.23	7.8	7.83	4.49	2.98	3.02	4.03
2	123.95	62.49	31.67	16.51	8.63	4.64	3.17	3.01	3.04	4.04	-
4	70.00	35.49	18.27	9.69	5.04	3.19	2.98	3.13	4.53	-	-
8	44.68	22.62	12.23	6.14	3.52	3.05	3.04	4.44	-	-	-
16	32.27	17.30	9.57	5.14	3.13	3.12	4.22	-	-	-	-
32	27.81	15.59	8.64	4.94	3.45	4.50	-	-	-	-	-
64	31.02	16.44	9.06	5.57	5.84	-	-	-	-	-	-
128	40.32	20.05	11.64	9.13	-	-	-	-	-	-	-
256	45.08	24.45	15.61	-	-	-	-	-	-	-	-
512	52.23	28.98	-	-	-	-	-	-	-	-	-
1 024	72.61	-	-	-	-	-	-	-	-	-	-

Matrix addition execution time (mili-seconds)

RESULTS II

	1	2	4	8	16	32	64	128	256	512	1 024
1	867 208	432 212	215 609	107 867	53 982	27 015	13 766	7 780	6 237	6 502	7 935
2	433 990	216 378	107 948	53 970	27 016	13 592	7 154	5 856	5 874	7 212	-
4	217 653	108 544	54 162	27 076	13 609	7 269	6 155	6 337	7 352	-	-
8	109 759	54 653	27 277	13 669	7 328	6 163	6 431	7 990	-	-	-
16	74 759	38 522	18 448	12 316	7 313	6 348	8 638	-	-	-	-
32	112 494	56 336	28 248	14 267	6 550	8 355	-	-	-	-	-
64	126 553	63 547	30 550	14 730	8 123	-	-	-	-	-	-
128	166 618	73 826	31 344	11 856	-	-	-	-	-	-	-
256	184 812	80 330	28 562	-	-	-	-	-	-	-	-
512	190 406	69 624	-	-	-	-	-	-	-	-	-
1024	194 297	-	-	-	-	-	-	-	-	-	-

Naive matrix multiplication execution time (mili-seconds)

- 256 and 512 are the best threadblock sizes

- Maximizing the Occupancy to improve the performance

- At least 32 threads per threadblock to fill the warps

- Threadblock column multiple of warp size (32)

- ↑ Number of threadblock-colums → ↑ performance

- Data reutilization and one row threadblock shape → global memory bottleneck (bad performance)

CONCLUSION AND FUTURE WORK

- Writing a efficient CUDA code as well as choosing a good global configuration is a complicated task for any programmer
- Maximizing the SM Occupancy improve the results (the resources are better exploited)
- Choosing Threadblocks to reduce the cache misses (fewer global memory accesses)
- The Fermi cache model is needed to understand the global memory accesses effects
- Currently, we are developing more sophisticate benchmark are been developed
- We are implementing the same experiments in OpenCL parallel model

REFERENCES

[1] DAVID B. KIRK AND WEN-MEI W. HWU Programming Massively Parallel Processors: A Hands-on Approach Morgan Kaufmann (Feb, 2010), ISBN: 978-0-12-381472-2.

[2] NVIDIA Whitepaper: NVIDIA's Next Generation CUDA Compute Architecture: Fermi. `http://www.nvidia.com/object/fermi_architecture.html`, Last visit: Nov, 2010, year=2010

[3] YURI TORRES, ARTURO GONZALEZ AND DIEGO R. LLANOS Simplyfing CUDA tuning techniques for Fermi. To appear in Proceedings of the Workshop on Exploitation of Hardware Accelerators (WEHA 2011)