Theoretical Performance Models for Graphics Processing Units

Taabish Jeshani, Marc Moreno Maza

University of Western Ontario

May 31, 2022

Contents

Plan

Background

General-purpose computing on graphics processing units

- GPGPU is commonly used in many research and industry fields across various disciplines.
- A major issue with GPGPU and programmers is the ability to identify performance bottlenecks accurately.
- GPUs are also increasing in power; Taking full advantage of this performance is sufficiently difficult.
- Thus, tools for annalyzing for both algorithms and code targeting GPUs are needed.

Performance models

- Performance models (like the MWP-CWP model) require the availability of machine-like code while algorithm models (like PRAM, MCM) do not.
- These analytical models give performance estimates (running time, memory consumption, etc.) which are more precise than those provided by algorithm models.

Challenges in designing a model of computation

Theoretical aspects

- GPU-like architectures introduces many machine parameters (like memory sizes, number of cores), and too many could lead to intractable calculations.
- GPU-like code depends also on program parameters (like number of threads per thread-block) which specify how the work is divided among the computing resources.

Practical aspects

- Analyzing parametric programs (with unknown machine and program parameters) require handling non-linear algebraic constraints.

Plan

MWP-CWP Background

- At a high level, MWP-CWP takes in several low-level performance metrics to estimate the execution time of a program.
- The basic idea behind MWP-CWP is estimating the cost of memory operations which in theory allows for the estimation of performance for parallel CUDA applications.
- Memory Warp Parallelism (MWP) uses the number of concurrently running threads and memory bandwidth consumption to estimate the cost of these memory operations.
- On the other hands, Computation Warp Parallelism (CWP) represents the amount of computation done by warps while one warp is waiting for memory values.

MWP-CWP goals

The MWP-CWP model (Sunpyo Hong & Hyesoon Kim, ISCA 2009)

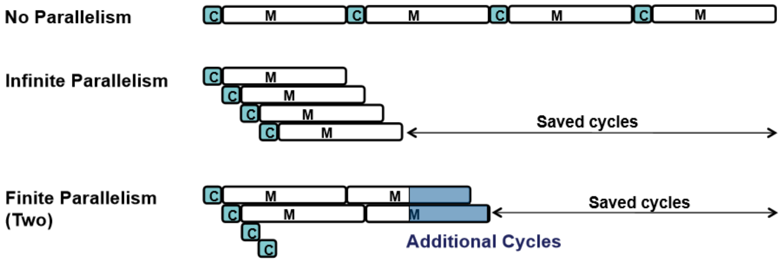
- aims at estimating T_{exec} as

$$T_{\mathrm{exec}} = T_{\mathrm{comp}} + T_{\mathrm{mem}} - T_{\mathrm{overlap}}$$

- determining T_{overlap} requires to understand whether computations hide memory latency or not
- thus requires hardware characteristics and instruction counts, thus access to the IR of a program.

Main observation of MWP-CWP

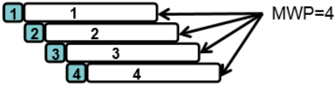
As we know, memory accesses can be overlapped between warps



Performance can be predicted by knowing the amount of *memory-level parallelism*.

Memory Warp Parallelism (MWP)

MWP is the maximum number of warps that can overlap memory accesses.

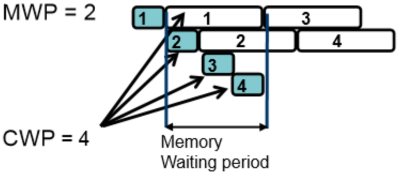


Four warps are overlapped during memory accesses

- Here, we $\mathrm{MWP} = 4$.
- MWP is determined by #Active SMs, #Active warps, Bandwidth, Types of memory accesses (Coalesced, Uncoalesced)

Computation Warp Parallelism (CWP)

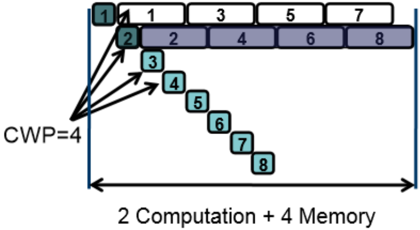
CWP is the number of warps that execute instructions during one memory access period plus one.



Here, we $\mathrm{CWP} = 4$.

Estimating the number of cycles of a kernel (1/2)

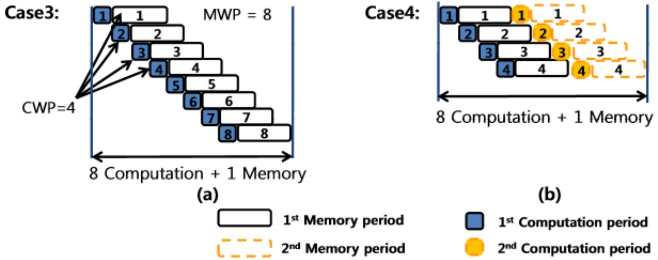
$MWP \leq CWP$



- Computation cycles are hidden by memory waiting periods
- Overall performance is dominated by the memory cycles

Estimating the number of cycles of a kernel (2/2)

MWP > CWP



- Memory accesses are mostly hidden due to high MWP
- Overall performance is dominated by the computation cycles

See also (Jaewoong Sim & Aniruddha Dasgupta & Hyesoon Kim & Richard Vuduc, PPoPP 12)

Determining MWP and CWP

Model Parameter	Obtained	Value
Mem_LD	Machine conf.	420
Departure_del_uncoal	Machine conf.	10
#Threads_per_block	Figure 12 Line 1	128
#Blocks	Figure 12 Line 1	80
#Active_blocks_per_SM	Occupancy [22]	5
#Active_SMs	Occupancy [22]	16
#Active_warps_per_SM	$128/32(Table\ 1) \times 5$	20
#Comp_insts	Figure 13	27
#Uncoal_Mem_insts	Figure 12 Lines 13, 14	6
#Coal_Mem_insts	Figure 12 Lines 13, 14	0
#Synch_insts	Figure 12 Lines 16, 21	$6 = 2 \times 3$
#Coal_per_mw	see Sec. 3.4.5	1
#Uncoal_per_mw	see Sec. 3.4.5	32
Load_bytes_per_warp	Figure 13 Lines 4, 6	128B = $4B \times 32$
Departure_delay	Equation (15)	320=32×10
Mem_L	Equations (10), (12)	730=$420 + (32 - 1) \times 10$
MWP_without_BW_full	Equation (16)	2.28 =$730/320$
BW_per_warp	Equation (7)	0.175GB/S = $\frac{1G \times 128B}{730}$
MWP_peak_BW	Equation (6)	28.57=$\frac{80GB/s}{0.175GB \times 16}$
MWP	Equation (5)	2.28=MIN(2.28, 28.57, 20)
Comp_cycles	Equation (19)	132 cycles= $4 \times (27 + 6)$
Mem_cycles	Equation (18)	$4380 = (730 \times 6)$
CWP_full	Equation (8)	34.18=$(4380 + 132)/132$
CWP	Equation (9)	20 = MIN(34.18, 20)
#Rep	Equation (21)	1 = $80/(16 \times 5)$
Exec_cycles_app	Equation (23)	$38450 = 4380 \times \frac{20}{2.28} + \frac{132}{6} \times (2.28 - 1)$
Synch_cost	Equation (26)	12288= $320 \times (2.28 - 1) \times 6 \times 5$
Final Time	Equation (27)	50738 =$38450 + 12288$

MWP-CWP: concluding remarks

- First analytic model that estimates the execution cycles for GPU
- Experimentally, quite successful on “average-size” kernels
- Recalibration for new hardware and in-depth analysis for new applications code
- Evaluation is fast
- The model only predicts execution time, no bottlenecks highlighting

GPUPerf

- Successor to MWP-CWP (built on top of MWP-CWP with more optimization techniques)
- Three main components: A frontend data collector, an analytical model, and a performance advisor.
- MWP-CWP falls short due the assumption that there is "enough intruction-level parallelism".

Threaded many-core memory (TMM) model

Ma, Agrawal and Chamberlain (2014) introduce the TMM model which retains many important characteristics of GPU-type architectures.

	Description
L	Time for a global memory access
P	Number of processors (cores)
C	Memory access width
Z	Size of fast private memory per core group
Q	Number of cores per core group
X	Hardware limit on number of threads per core

Table: Machine parameters of the TMM model

- In TMM analysis, the running time of algorithm is estimated by choosing the maximum quantity among the work, span and amount of memory accesses. No Graham-Brent theorem-like is provided.
- Such running time estimates depend on the machine parameters.

The MCM model

(S. A, Haque, N, Xie, M., 2013) proposes a many-core machine (MCM) model which aims at

- tuning program parameters to minimize parallelism overheads of algorithms targeting GPU-like architectures as well as
- comparing different algorithms independently of the value of machine parameters of the targeted hardware device.

In the design of this model, we insist on the following features:

- Two-level DAG programs
- Parallelism overhead

Characteristics of the abstract many-core machines (1/2)

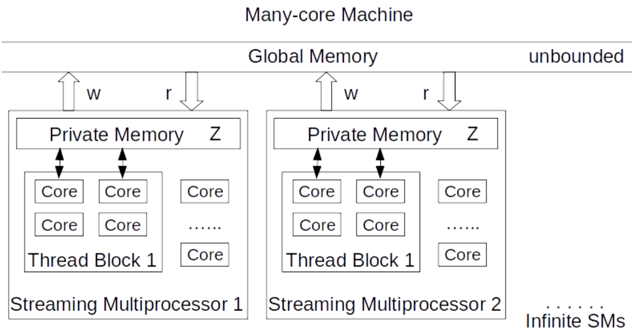


Figure: A many-core machine

- It has a global memory with high latency, while private memories have low latency.

Characteristics of the abstract many-core machines (2/2)

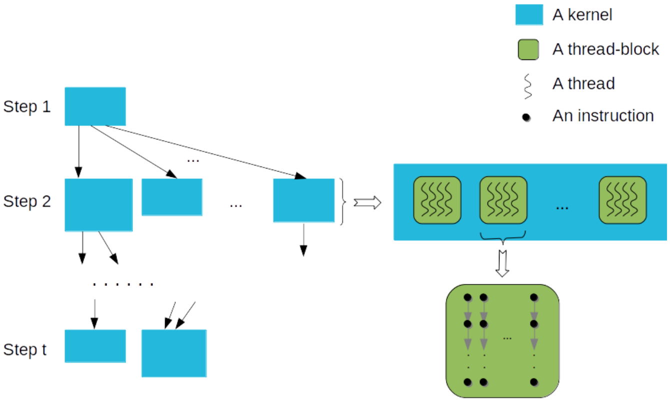


Figure: Overview of a many-core machine program, also called *kernel DAG*

Machine parameters and complexity measures

Machine parameters

- Z: Private memory size of any SM
- U: Data transfer time

Complexity measures

- **work** $W(\mathcal{P})$ is the total work of all its kernels;
- **span** $S(\mathcal{P})$ is the longest path, counting the weight (span) of each vertex (kernel), in the kernel DAG;
- **parallelism overhead** $O(\mathcal{P})$ is the total parallelism overhead (i.e. data transfer time) of all its kernels.

Plan

Matrix Multiplication

Kernel Input				Elapsed Time (ms)	execution cycles
1024,	4	*	4	101.15	1410580510
1024,	8	*	8	78.11	218486863
1024,	16	*	16	75.79	66568741
1024,	32	*	32	75.15	16295843
512,	4	*	4	71.98	109441566
512,	8	*	8	70.24	19019599
512,	16	*	16	68.76	4381830
512,	32	*	32	69.46	1935861
256,	4	*	4	68.52	9526430
256,	8	*	8	68.69	1253278
256,	16	*	16	68.60	829293
256,	32	*	32	67.39	177923

2D Convolution

Kernel Input				Elapsed Time (ms)	execution cycles
4096,	2	*	2	79.38	109913262
4096,	4	*	4	74.28	55541774
4096,	8	*	8	71.30	28262428
4096,	16	*	16	72.26	24299618
4096,	32	*	32	70.99	22364518
2048,	2	*	2	69.59	27478336
2048,	4	*	4	68.63	13885454
2048,	8	*	8	69.57	7065628
2048,	16	*	16	68.79	6074978
2048,	32	*	32	68.71	5591398
1024,	2	*	2	70.21	6869605
1024,	4	*	4	71.78	3471374
1024,	8	*	8	72.08	1766428
1024,	16	*	16	68.26	1518818
1024,	32	*	32	68.61	1397939

SYRK (Symmetric rank-k update)

Kernel Input				Elapsed Time (ms)	execution cycles
1024,	2	*	2	181.57	883895500801
1024,	4	*	4	126.87	440367566849
1024,	8	*	8	108.14	226630428675
1024,	16	*	16	114.32	206762846988
1024,	32	*	32	157.07	211636194456
512,	2	*	2	88.13	55726686209
512,	4	*	4	75.51	27665691649
512,	8	*	8	72.11	14236023299
512,	16	*	16	73.62	14840789511
512,	32	*	32	77.77	16352705031
256,	2	*	2	69.26	3543429121
256,	4	*	4	69.91	1746976001
256,	8	*	8	68.22	898719361
256,	16	*	16	69.49	936566401
256,	32	*	32	69.53	1031184001

Plan

Future Work

- Saturated vs Unsaturated kernels
- Taking a deeper dive into more performance models focused on program analysis
- KLARAPTOR