Makespan computation for GPU threads running on a single streaming multiprocessor

CISTER - Research Centre in
Real-Time & Embedded Computing Systems

Kostiantyn Berezovskyi
CISTER/INESC-TEC, ISEP
Porto, Portugal

Konstantinos Bletsas
CISTER/INESC-TEC, ISEP
Porto, Portugal

Björn Andersson
SEI CMU
Pittsburgh, USA

Supported by the REGAIN project, ref. FCOMP-01-0124-FEDER-020447,
co-funded by National Funds through the FCT-MCTES ERDF through COMPETE under PhD grant SFRH/BD/82069/2011

Stream Processing

Streams

Collection of data.

All data is expressed in streams.

Kernels

Series of operations.

Input: streams.

Output: streams.

Why Streams?

Data parallelism

Stream elements can be processed at once.

Task parallelism

Pipeline.

© nvidia.com

GPU software application

Large data collections.

Data parallelism.

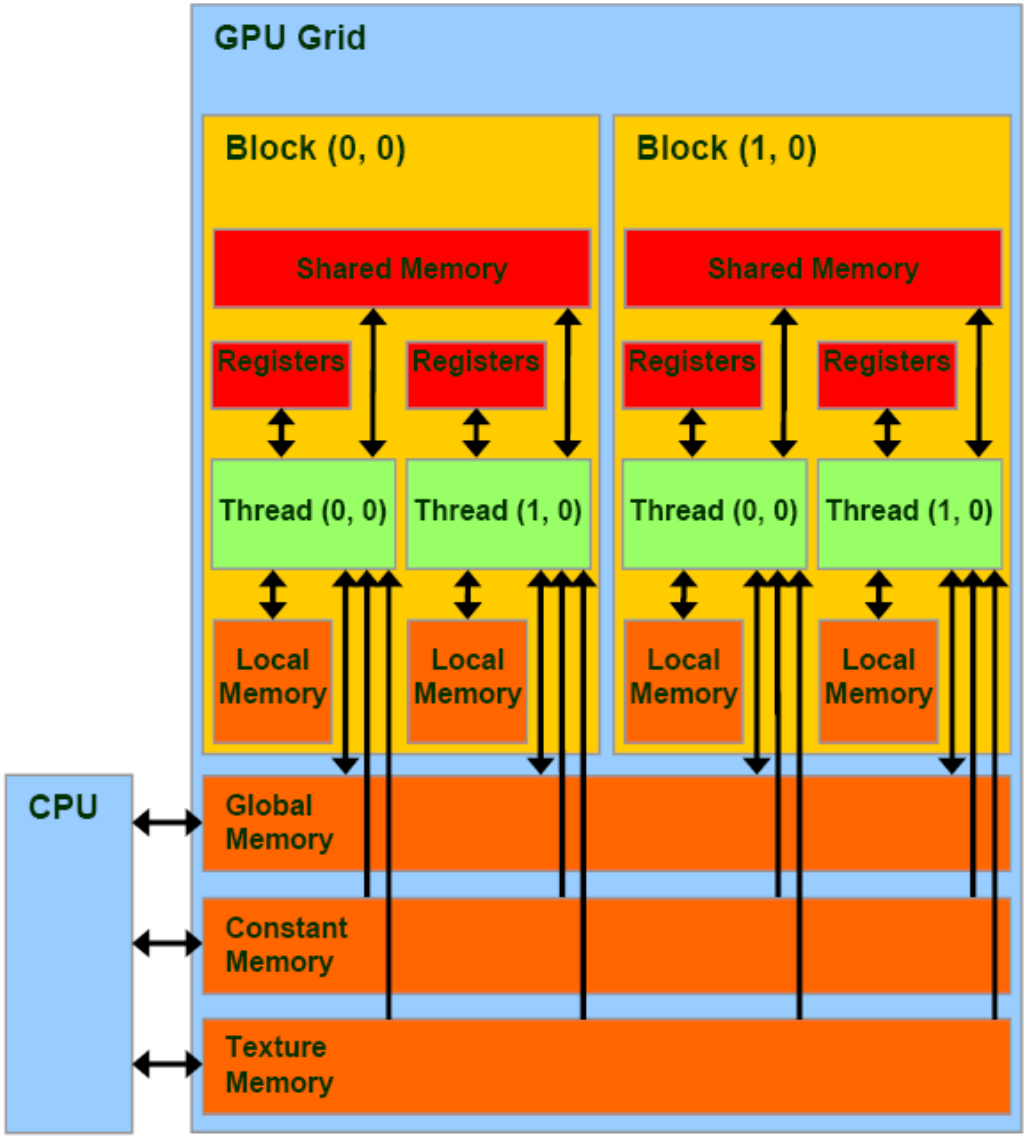
High arithmetic intensity.

Minimal dependency between data elements.

Application areas

© nvidia.com

GPU as a co-processor



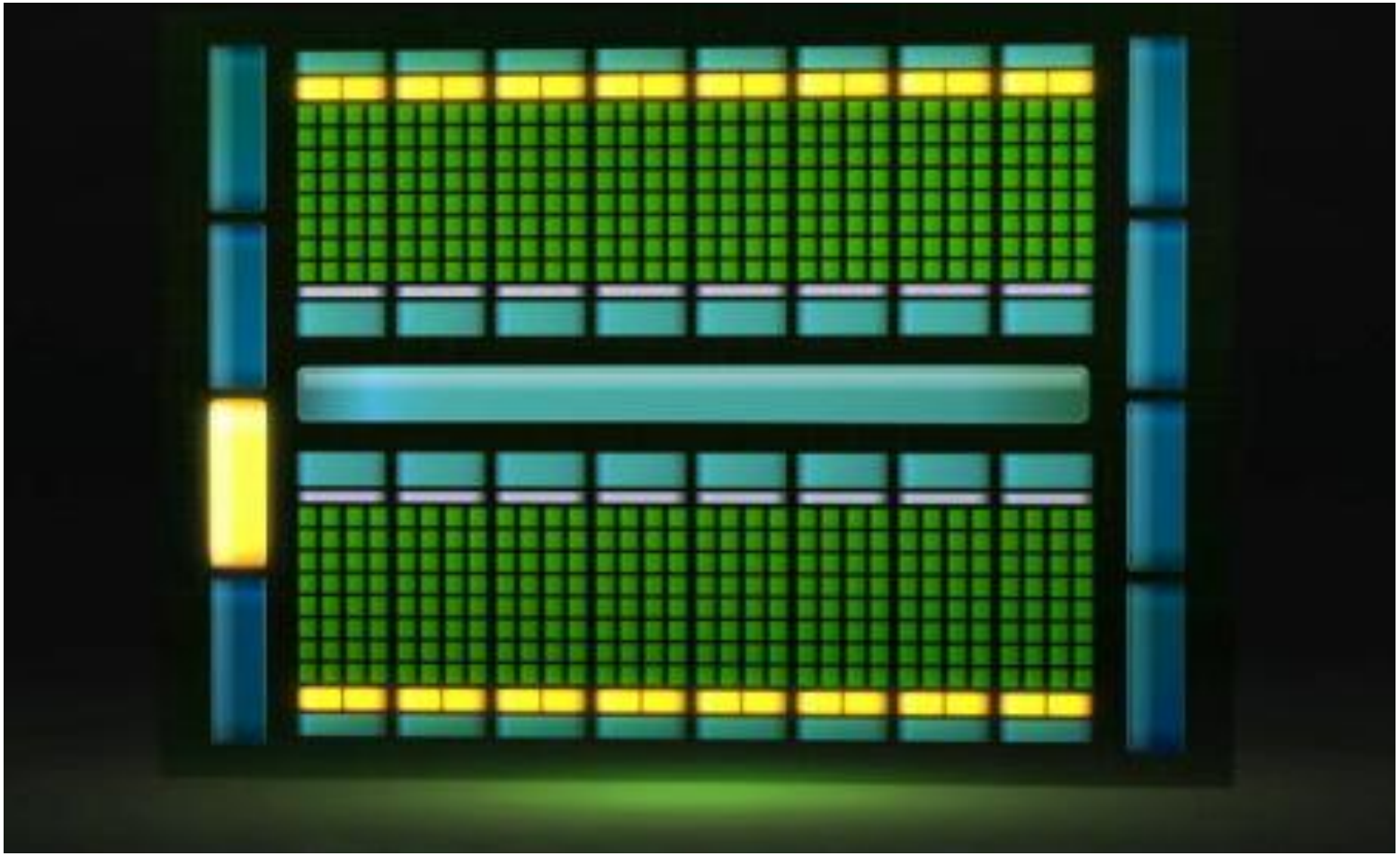
Related work

G. Elliott and J. Anderson.
Real-time multiprocessor systems with GPUs.
18th International Conference on Real-Time and Network Systems (RTNS), 2010.

S. Kato, K. Lakshmanan, A. Kumar, Y. Ishikawa, and R. Rajkumar.
Rgem: A responsive GPGPU execution model for runtime engines.
32nd IEEE Real-Time Systems Symposium (RTSS), 2011.

R. Mangharam and A. A. Saba.
Anytime algorithms for GPU architectures.
32nd IEEE Real-Time Systems Symposium (RTSS), 2011.

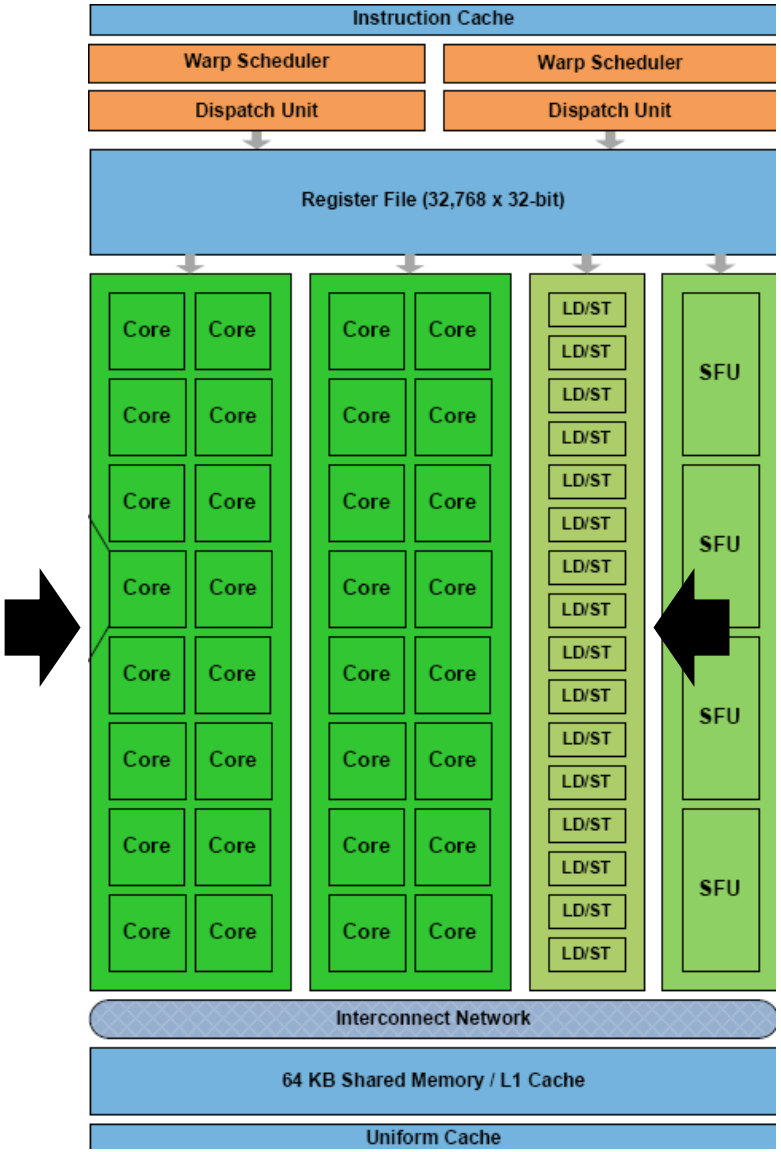
Fermi Architecture



© nvidia.com

16 streaming multiprocessors

Streaming multiprocessor

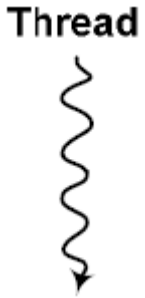


CUDA cores

load/store units

Fermi Streaming Multiprocessor (SM)

Computation entities



thread ID

© mohaircarpets.co.za

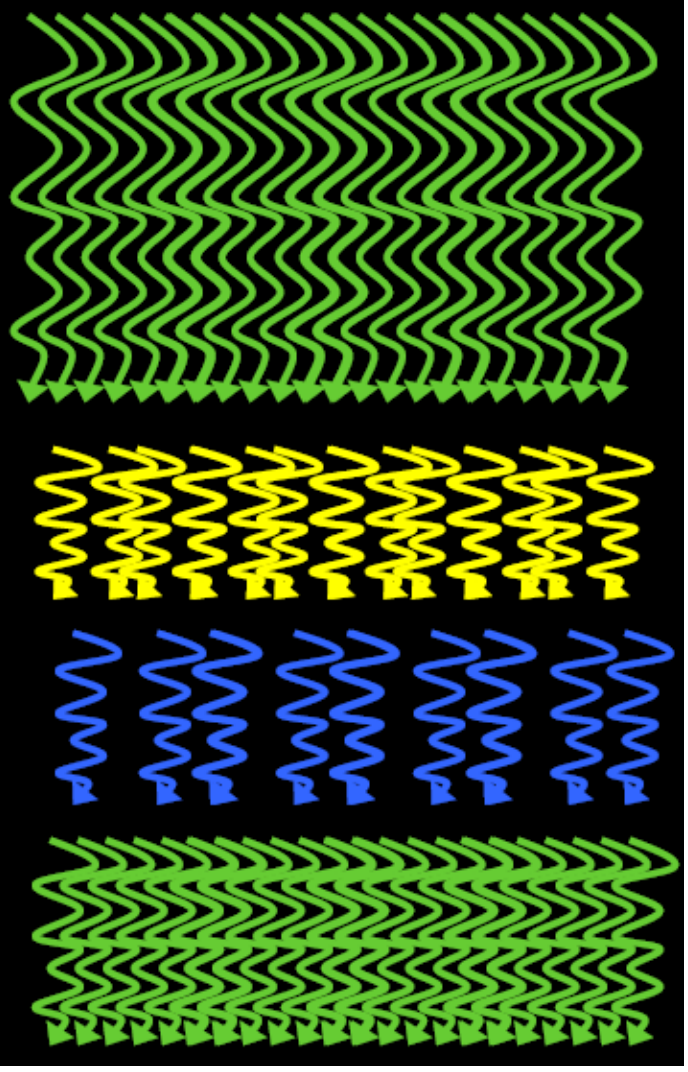
Warp

a group of threads with consecutive IDs; all should be executed in parallel.

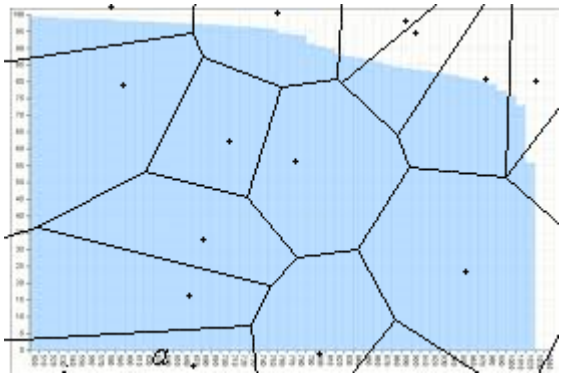
© nvidia.com

Control flow divergence

```
__global__ void kv(int* x, int* y)
{
  int i = threadIdx.x +
    blockDim.x * blockIdx.x;
  int t;
  bool b = f(x[i]);
  if (b)
  {                  // g(x)
    t = g(x[i]);
  }
  else
  {                  // h(x)
    t = h(x[i]));
  }
  y[i] = t;
}
```

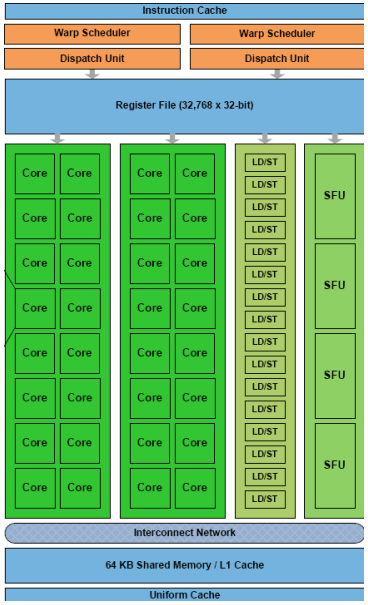


How does it work?



```
float md =  (x-x1)^2+(y-y1)^2  ;          //minimal distance
int mdp = 1;                              //minimal distance point
for (int i=2; i<=N; i++)
        if (  (x-xi)^2+(y-yi)^2  < md){
                md =  (x-xi)^2+(y-yi)^2  ;
                mdp = i;
        }
```

Parallel Thread Execution (PTX)



© nvidia.com

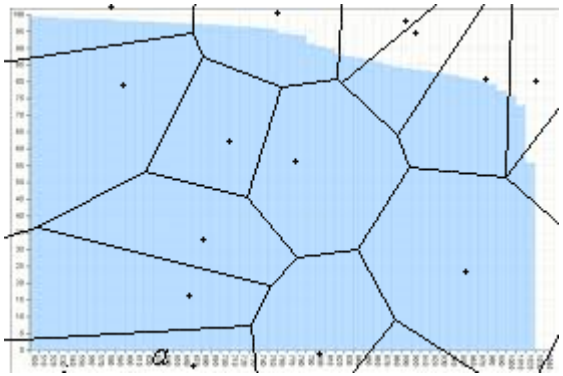
CUDA cores

load/store units

PTX Assembly Code

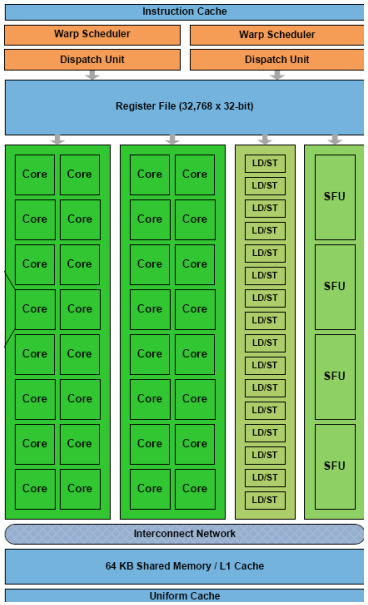
```
        mov.u32     $r0,    N_addr;          // N                                  L
        mov.f32     $f1,    x_addr;          // x                                  L
        mov.f32     $f2,    y_addr;          // y                                  L
        mov.f32     $f3,    x1_addr          // x1                                 L
        mov.f32     $f4,    y1_addr;         // y1                                 L
        sub.f32     $f5,    $f1,    $f3;     // (x-x1)                             C
        mul.f32     $f6,    $f5,    $f5;     // (x-x1)^2                           C
        sub.f32     $f7,    $f2,    $f4;     // (y-y1)                             C
        fma.f32     $f8,    $f7,    $f7,    $f6;  // (x-x1)^2+(y-y1)^2             C
        mov.f32     $f0,    $f8;             // md = (x-x1)^2+(y-y1)^2             C
        mov.u32     $r1,    1;               // int mdp = 1;                       C
        mov.u32     $r2,    2;               // for (int i=2; i<=N; i++)           C
Loop:   setp.gt.u32 p,      $r2,    $r0;     // for (int i=2; i<=N; i++)           C
@p      bra Done;                            // for (int i=2; i<=N; i++)           C
        mov.f32     $f3,    xi_addr;         // xi                                 L
        mov.f32     $f4,    yi_addr;         // yi                                 L
        sub.f32     $f5,    $f1,    $f3;     // (x-xi)                             C
        mul.f32     $f6,    $f5,    $f5;     // (x-xi)^2                           C
        sub.f32     $f7,    $f2,    $f4;     // (y-yi)                             C
        fma.f32     $f8,    $f7,    $f7,    $f6;  // (x-xi)^2+(y-yi)^2             C
        setp.ge.f32 q,      $f8,    $f0;     // if ( (x-xi)^2+(y-yi)^2 < md)       C
@q      bra If;                                                                    C
        mov.f32     $f0,    $f8;             // md = (x-xi)^2+(y-yi)^2 ;           C
        mov.u32     $r1,    $r2;             // mdp = i;                           C
If:                                          // if ( (x-xi)^2+(y-yi)^2 < md)
        add.u32     $r2,    $r2,    1;       // for (int i=2; i<=N; i++)           C
        bra Loop;                                                                  C
Done:
```

Kernel instruction string



```
float md =   (x-x1)^2+(y-y1)^2   ;          //minimal distance
int mdp = 1;                                //minimal distance point
for (int i=2; i<=N; i++)
        if (  (x-xi)^2+(y-yi)^2  < md){
                md =   (x-xi)^2+(y-yi)^2  ;
                mdp = i;
        }
```

L L L L L C C C C C C C C C L L C C C C C C C C



© nvidia.com

CUDA cores (C)

load/store units (L)

Related work

An Adaptive Performance Modeling Tool for GPU Architectures

Sara S. Baghsorkhi, Matthieu Delahaye, Sanjay J. Patel, Willian D. Gropp, Wen-mei W. Hwu

15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), 2010.

A Memory-level and Thread-level Parallelism Aware GPU Architecture Performance Analytical Model,

Sunpyo Hong and Hyesoon Kim

36th International Symposium on Computer Architecture (ISCA-36), 2009.

Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA

Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen-mei W. Hwu

13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2008.

The makespan

Makespan

Clock Cycle	1	2	3	4	5	6	7	8	9
Warp 1	C	L		L	C	L		L	
Warp 2		C	L		L	C	L		L

Worst – case

Makespan

Clock Cycle	1	2	3	4	5	6	7	8	9	10
Warp 1	C	L	L	C	L	L				
Warp 2		C		L			L	C	L	L

Assumptions

Every instruction takes single clock cycle.

Cache

A warp on a single SM

32-bit float instruction

Cache hit

1 clock cycle

Cache miss

400-600 clock cycles

Assumptions

Every instruction takes single clock cycle.

No cache misses.

Scheduling Policy

C L L C L L 2 warps

Clock Cycle	1	2	3	4	5	6	7	8	9	10
Warp 1	C	L			L	C	L		L	
Warp 2			C	L		L	C	L		L

Non-work-conserving schedule

Clock Cycle	1	2	3	4	5	6	7	8	9
Warp 1	C	L		L	C	L		L	
Warp 2		C	L		L	C	L		L

Work-conserving schedule

Assumptions

Every instruction takes single clock cycle.

No cache misses.

Work-conserving scheduling.

The Pessimistic Method

C L L C L L

Clock Cycle	1	2	3	4	5	6	7	8	9
Warp 1	C	L		L	C	L		L	
Warp 2		C	L		L	C	L		L

L L L L C C

Clock Cycle	1	2	3	4	5	6	7	8	9	10	11
Warp 1	L		L		L		L	C		C	
Warp 2		L		L		L		L	C		C

$$M = M^L + M^C$$

Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12
Warp 1	L		L		L		L		C		C	
Warp 2		L		L		L		L		C		C

M^C a makespan for C-substring

M^L a makespan for L-substring

Optimization Problem

Decision variables

- Usage of hardware units at particular clock cycle
- The pessimistic makespan – an upper bound

Objective function

- Maximize the makespan

Constraints

- Capacity constraints
- Precedence constraints
- Work-conserving constraints

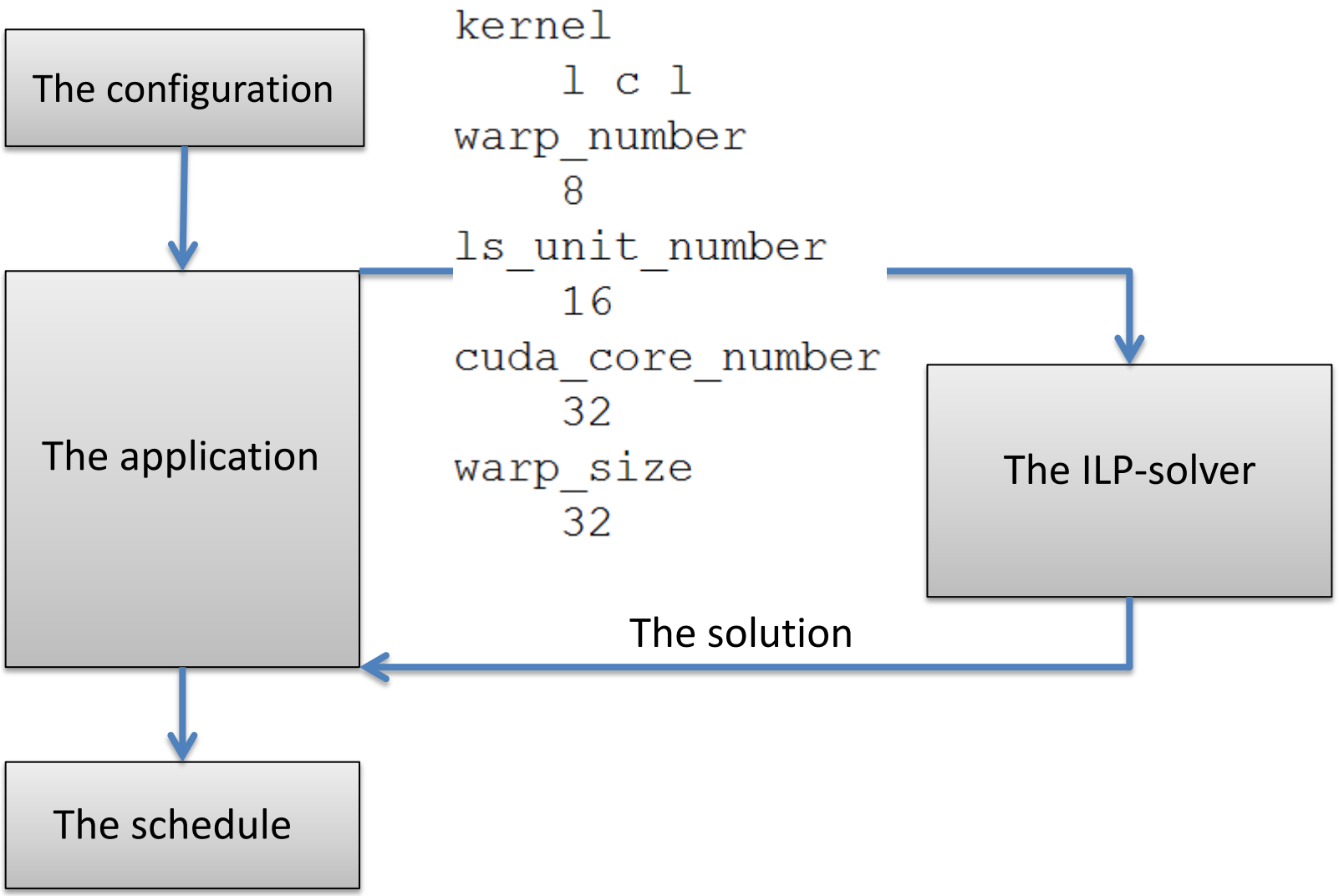
Integer Linear Problem (ILP)

Optimization Problem

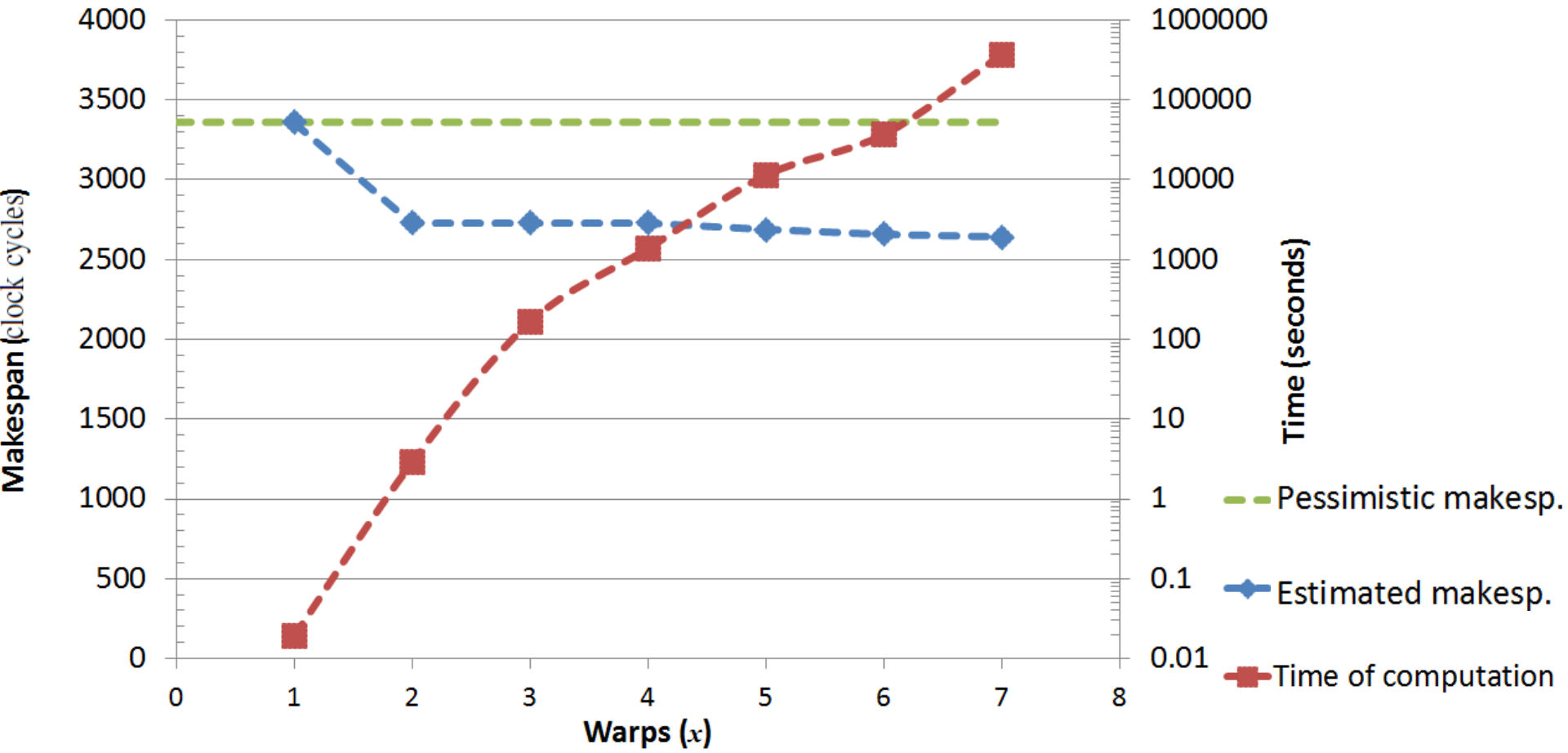
maximize $\sum_{t=1}^{T}(t \times (LS_{W,I,t} + CC_{W,I,t}))$ subject to

iterated variables	expression for constraint	number of constraints
$\forall t$	$\sum_{w=1}^{W}\sum_{i=1}^{I} LS_{w,i,t} \leq \sigma_L$	T
$\forall t$	$\sum_{w=1}^{W}\sum_{i=1}^{I} CC_{w,i,t} \leq \sigma_C$	T
$\forall w = 1..(W-1)$	$\sum_{t=1}^{T}(t \times (LS_{w,I,t} + CC_{w,I,t})) \leq \sum_{t=1}^{T}(t \times (LS_{W,I,t} + CC_{W,I,t}))$	$W-1$
$\forall w, t$	$\sum_{i=1}^{I} LS_{w,i,t} \leq 1$	$W \times T$
$\forall w, t$	$\sum_{i=1}^{I} CC_{w,i,t} \leq 1$	$W \times T$
$\forall w, i$	$\sum_{t=1}^{T} LS_{w,i,t} = IL_i$	$W \times I$
$\forall w, i$	$\sum_{t=1}^{T} CC_{w,i,t} = IC_i$	$W \times I$
$\forall w, i = 1..(I-1)$	$\sum_{t=1}^{T}(t \times (LS_{w,i,t} + CC_{w,i,t})) < \sum_{t=1}^{T}(t \times (LS_{w,i+1,t} + CC_{w,i+1,t}))$	$W \times (I-1)$
$\forall t$	$E_t \geq 1 - \sigma_L + \sum_{w'=1}^{W}\sum_{i=1}^{I} LS_{w',i,t}$	T
$\forall t$	$E_t \times \sigma_L \leq \sum_{w'=1}^{W}\sum_{i=1}^{I} LS_{w',i,t}$	T
$\forall w, i, t$	$\frac{1}{2}(TL_{w,i,t} + E_t) \leq TL_{w,i,t} \vee E_t \leq TL_{w,i,t} + E_t$	$W \times I \times T$
$\forall t$	$G_t \geq 1 - \sigma_C + \sum_{w'=1}^{W}\sum_{i=1}^{I} CC_{w',i,t}$	T
$\forall t$	$G_t \times \sigma_C \leq \sum_{w'=1}^{W}\sum_{i=1}^{I} CC_{w',i,t}$	T
$\forall w, i, t$	$\frac{1}{2}(TC_{w,i,t} + G_t) \leq TC_{w,i,t} \vee G_t \leq TC_{w,i,t} + G_t$	$W \times I \times T$

The Application



Resolving the issue of tractability



Fermi

48 warps

$$W = 420$$

Future Work

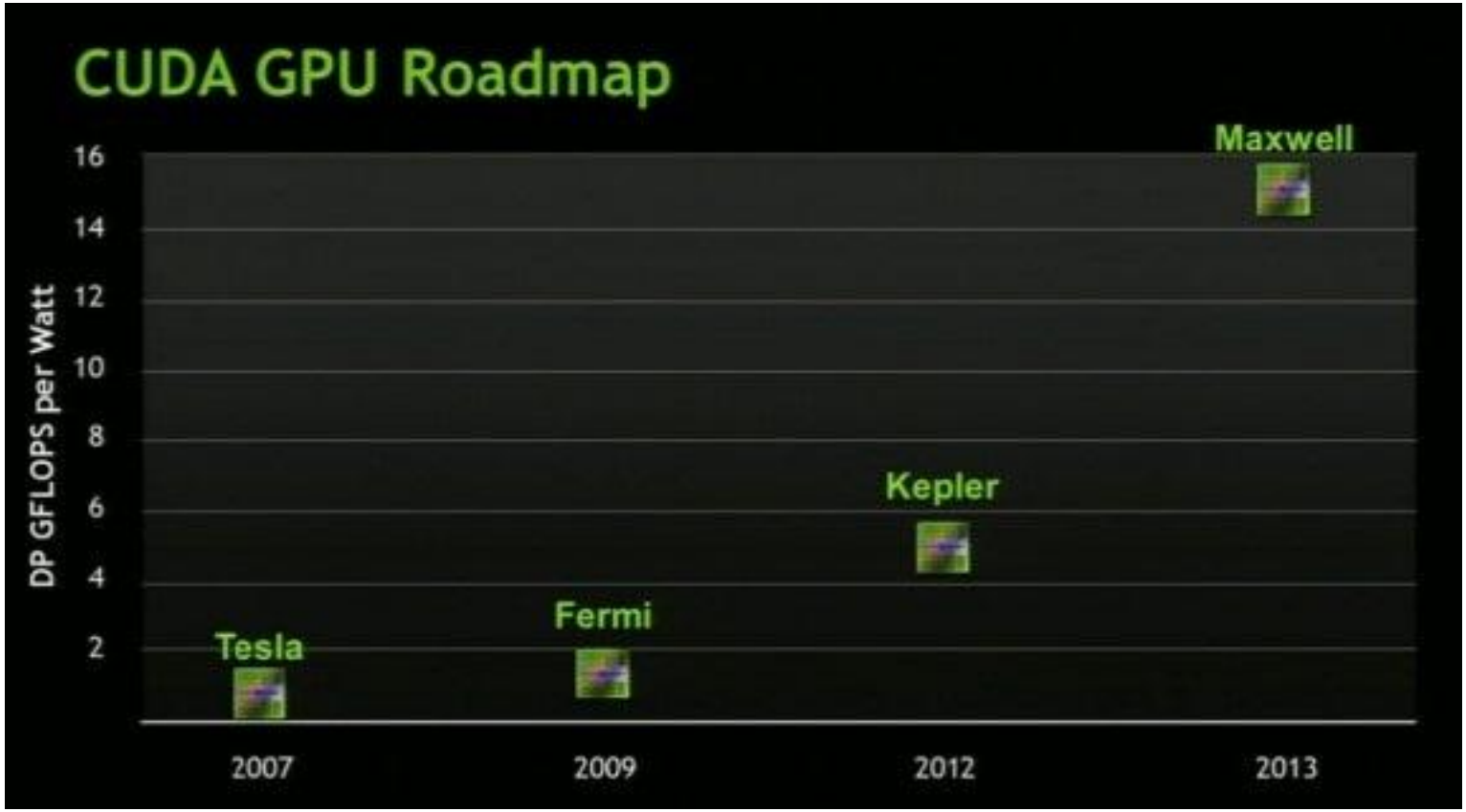
Relaxation of the “no cache miss assumption”.

Targeting the “whole” GPU.

Future Work

Fermi
48 warps

$$W = 420,$$



Questions?

Thank You!