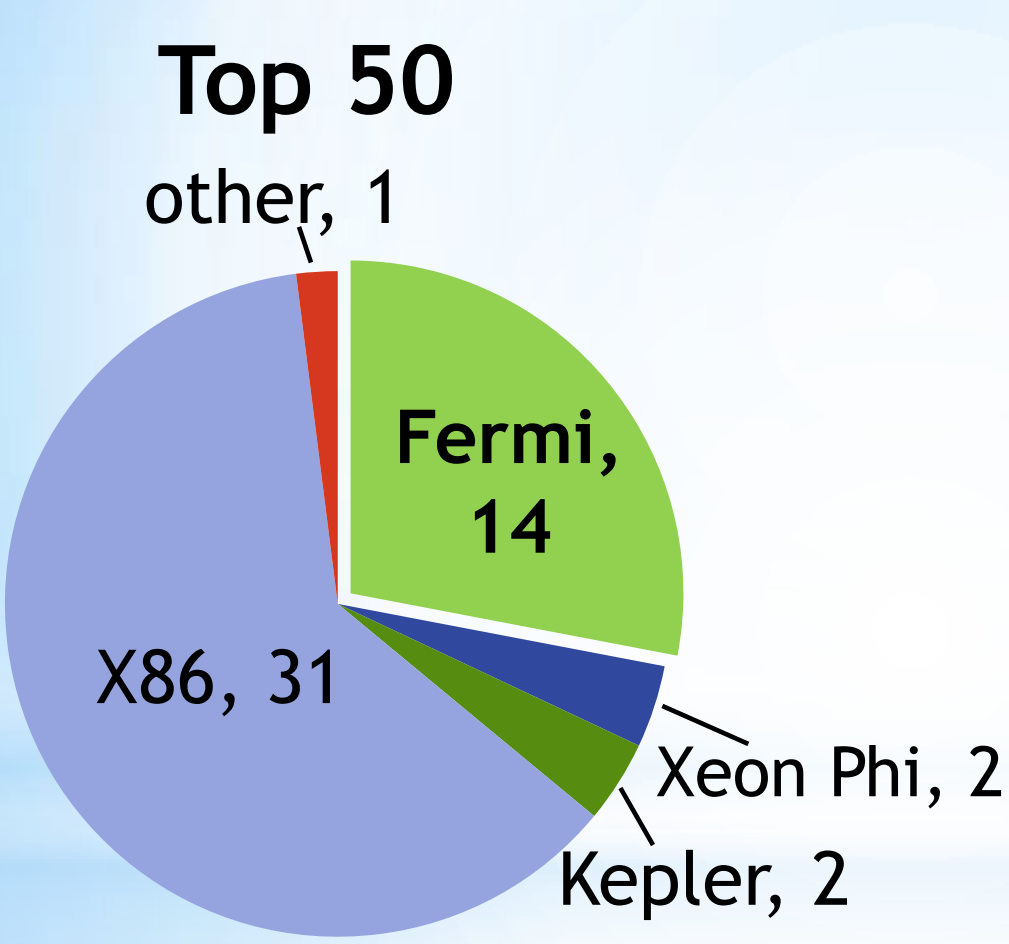
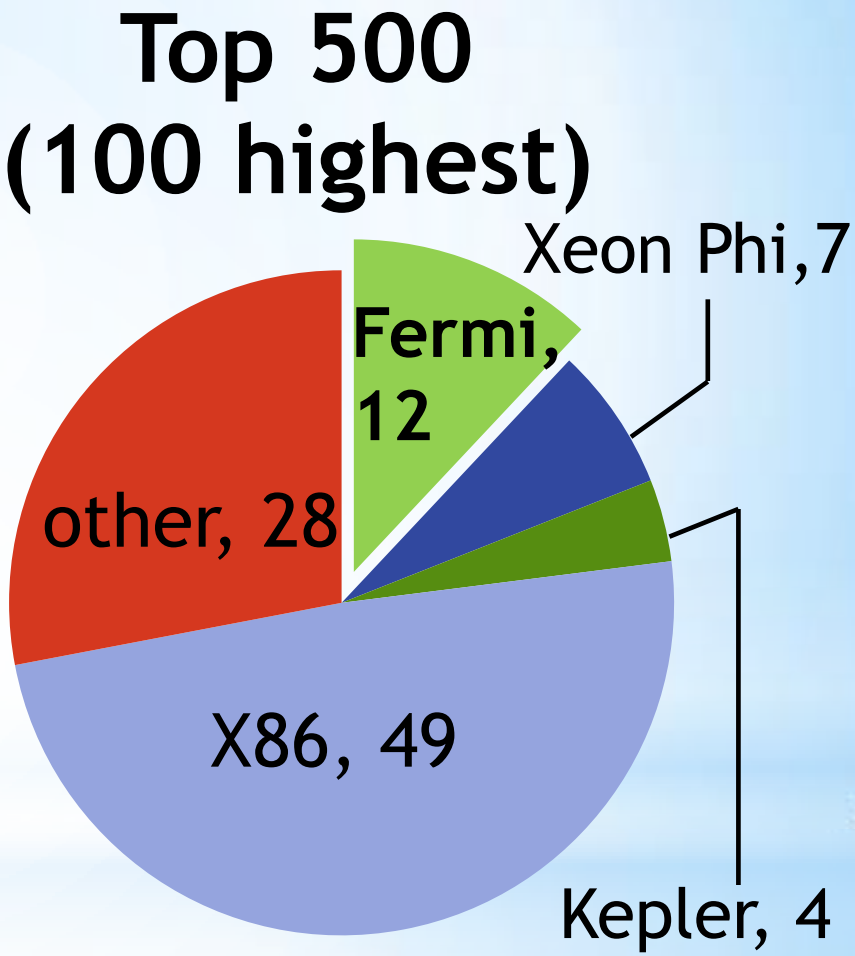
МОСКОВСКИЙ СУПЕРКОМПЬЮТЕРНЫЙ ФОРУМ 2013

Графические процессоры в суперкомпьютерных системах

Корнеев В.В., *д.т.н.*, Павлухин П. В., Шевченко И.В.,
ФГУП НИИ "Квант"

Ускорители в суперкомпьютерах





Семейство GPU Fermi занимает основную долю среди систем, оснащенных ускорителями

Микроархитектура GPU

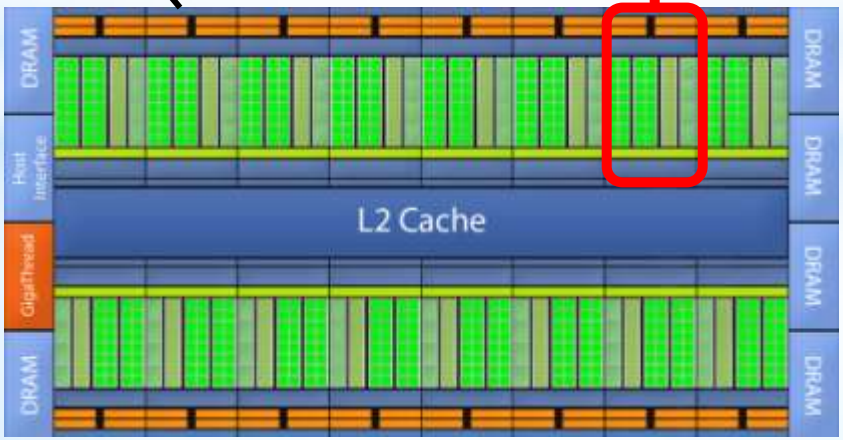
Наряду с тредовым параллелизмом (Thread Level Parallelism, *TLP*) в GPU важную роль играет и ***внутри***тредовый параллелизм (Instruction Level Parallelism, *ILP*). Для тонкой оптимизации приложений с учетом этих двух уровней параллельности необходимо достаточно глубокое знание микроархитектуры GPU, основу которой составляют два конвейера – **вычислительный (ALU)** и управляющий доступом в память **(Load/Store)**. Детали их работы, а именно:

- время исполнения инструкций разных типов
- число поддерживаемых незавершенных обращений в память,

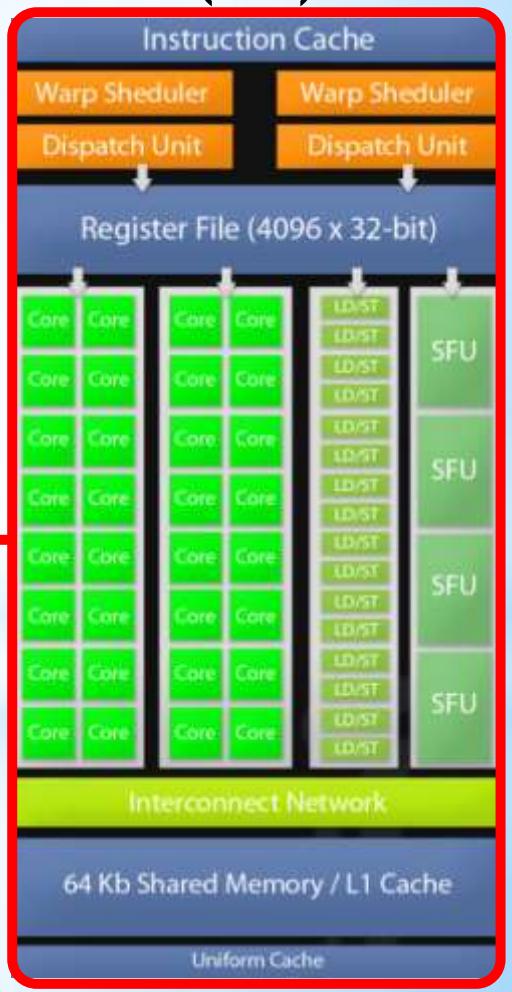
были изучены в нижеследующих тестах.

Архитектура GPU Fermi
(compute capability 2.0)

Tesla X2090: 16 SMs,
665 GFLOPS (DP)



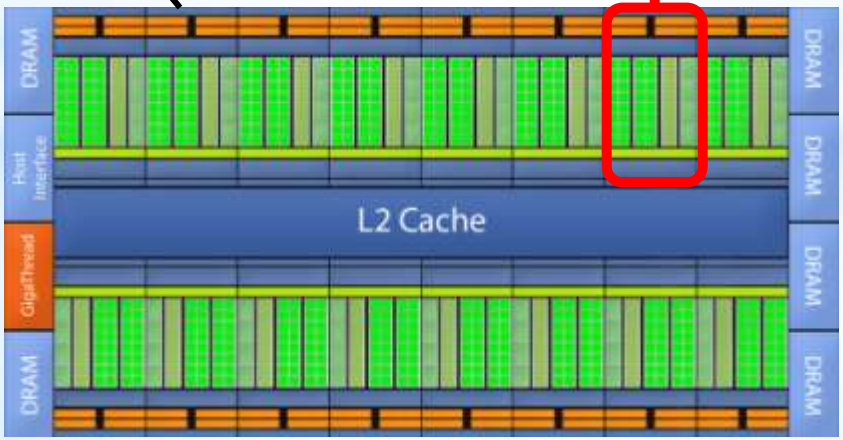
Stream Multiprocessor (SM)



Архитектура GPU Fermi
(compute capability 2.0)

32 треда, выполняющихся синхронно, объединяются в **варп**. Каждый SM может работать с 48 варпами (1536 тредов), «мгновенно» переключаясь между ними.

Tesla X2090: 16 SMs,
665 GFLOPS (DP)



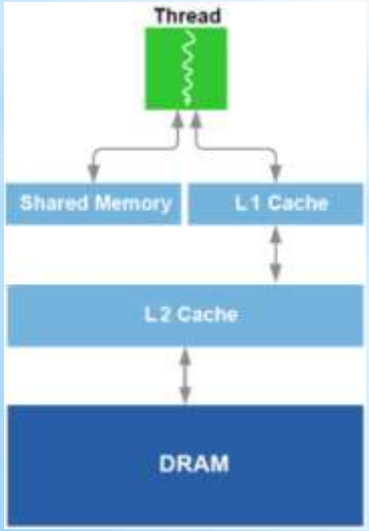
Stream Multiprocessor (SM)

Архитектура GPU Fermi
(compute capability 2.0)

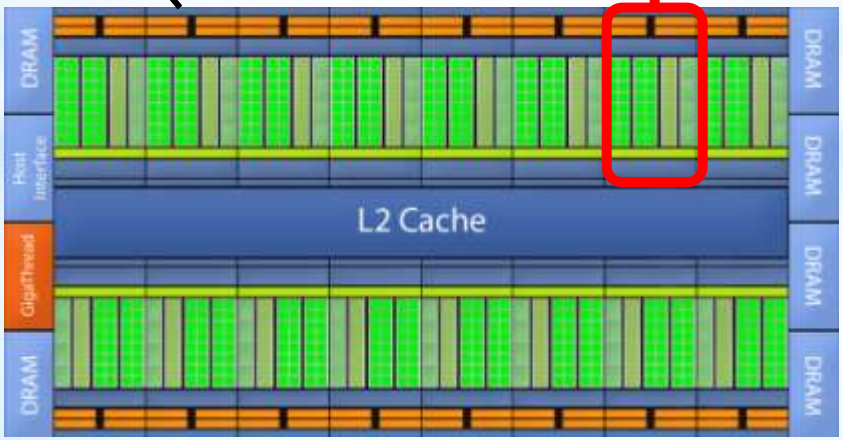
32 треда, выполняющихся синхронно, объединяются в **варп**. Каждый SM может работать с 48 варпами (1536 тредов), «мгновенно» переключаясь между ними.

Tesla X2090: 16 SMs, 665 GFLOPS (DP)

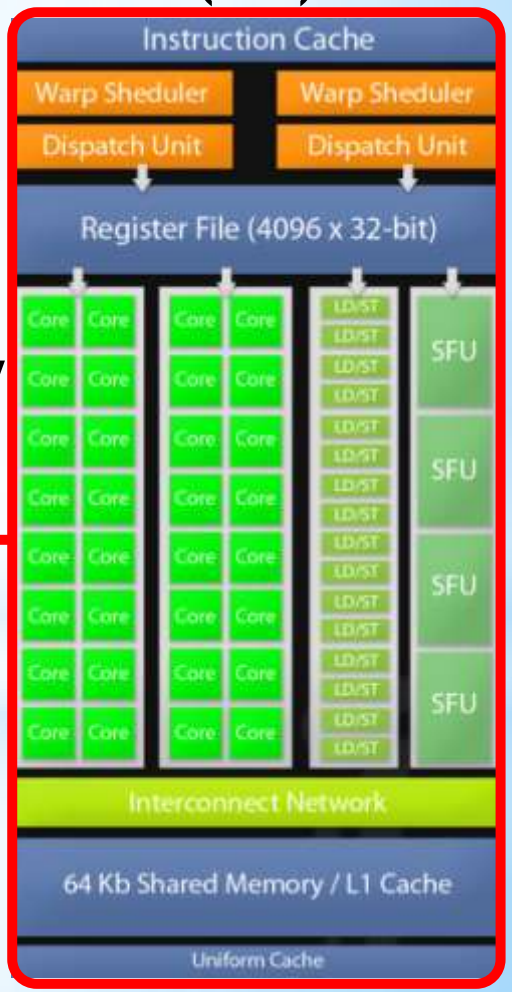
Иерархия памяти



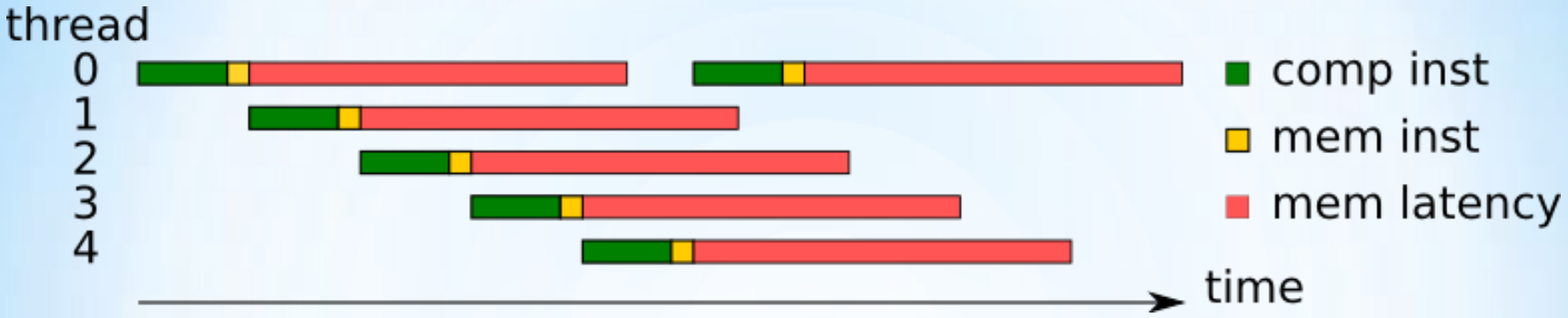
Каждый SM содержит:
- Register File
- L1 cache/Shared Memory
- LD/ST Unit
- 32-core ALU



Stream Multiprocessor (SM)



Архитектура GPU Fermi



Эффективность работы – за счет совмещения вычислительной работы в одних варпах и ожидания завершения обращений к памяти в других

Доступные языки программирования:

- **CUDA C** – высокоуровневый язык;
- **PTX** – «промежуточный» язык, псевдоассемблер;
- **AsFermi*** – ассемблер для Fermi (unofficial);

** http://code.google.com/p/asfermi/*

AsFermi

Формат **cuobjdump** для исходного кода:

- Регистры R1 – R63;
- Инструкции *ADD, *MUL, *FMA, LD, ST, MEMBAR;
- Спецрегистры (SR_ClockLo);

Основной прием для написания тестов:

```
S2R R0, SR_ClockLo;  //аналог clock64() в CUDA C
INST_1 …
INST_2 …
…
INST_N …
S2R R1, SR_ClockLo;
```

Delta = **R1 - R0** – число тактов, затраченных на выполнение инструкций

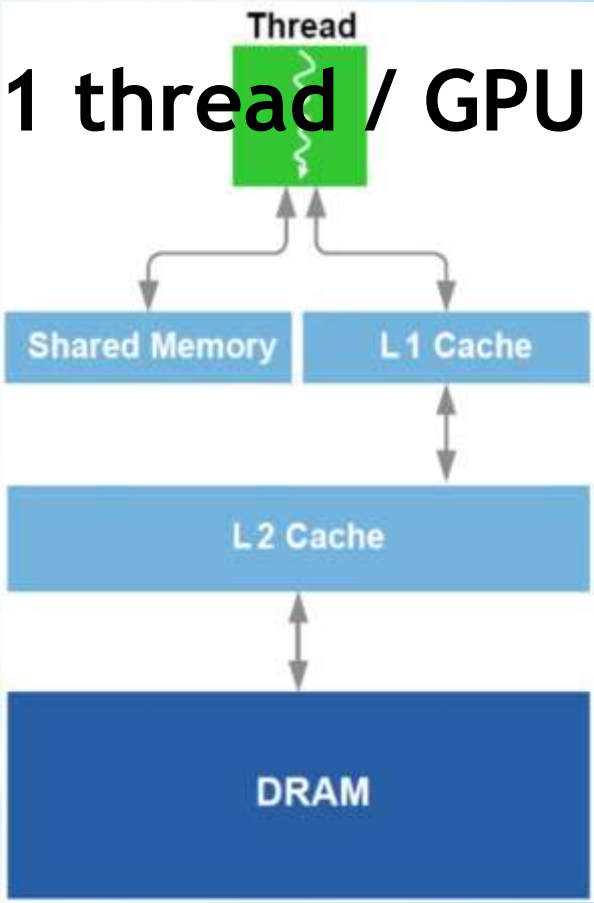
Задержки при обращении к памяти

Global memory (DRAM):
LD (L1 &L2 miss) - ~510 cycles
LD (L1 hit) - **17** cycles
LD (~~L1~~; L2 miss) - ~ 495 cycles
LD (~~L1~~; L2 hit) - ~ 274 cycles

Shared memory:
LDS – **26** cycles
STS – **30** cycles

Shared memory **медленнее** L1!

MEMBAR:
per SM – 12 cycles
Per GPU – 30 cycles

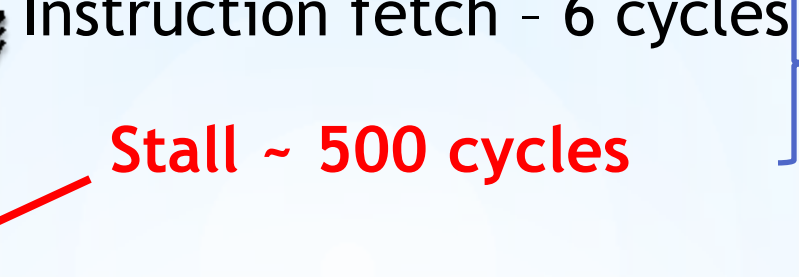


__syncthreads() – 34 cycles (1 thread/SM),
~64 cycles (512 threads/SM)

LD/ST конвейер

32 threads (1 warp) / SM:

```
LD.E [R8+0x0], R0
LD.E [R8+0x4], R1
LD.E [R8+0x8], R2
LD.E [R8+0xC], R3
LD.E [R8+0x10],R4
LD.E [R8+0x14],R5
```



до 5 подряд идущих незавершенных обращений к памяти на варп!

LD/ST конвейер

32 threads (1 warp) / SM:

```
LD.E [R8+0x0], R0
LD.E [R8+0x4], R1
LD.E [R8+0x8], R2
LD.E [R8+0xC], R3
LD.E [R8+0x10],R4
IADD R5, R6, R7
```

Instruction fetch – 6 cycles

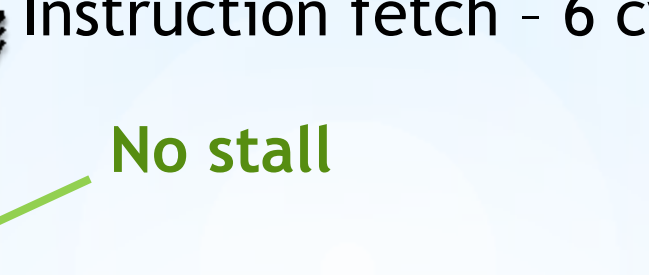
No stall

до 5 подряд идущих незавершенных обращений к памяти на варп!

LD/ST конвейер

32 threads (1 warp) / SM:

```
LD.E [R8+0x0], R0
LD.E [R8+0x4], R1
LD.E [R8+0x8], R2
LD.E [R8+0xC], R3
LD.E [R8+0x10],R4
IADD R5, R6, R7
```



до 5 подряд идущих незавершенных обращений к памяти на варп!

А если увеличить кол-во варпов на SM?

LD/ST конвейер

32 threads (1 warp) / SM:

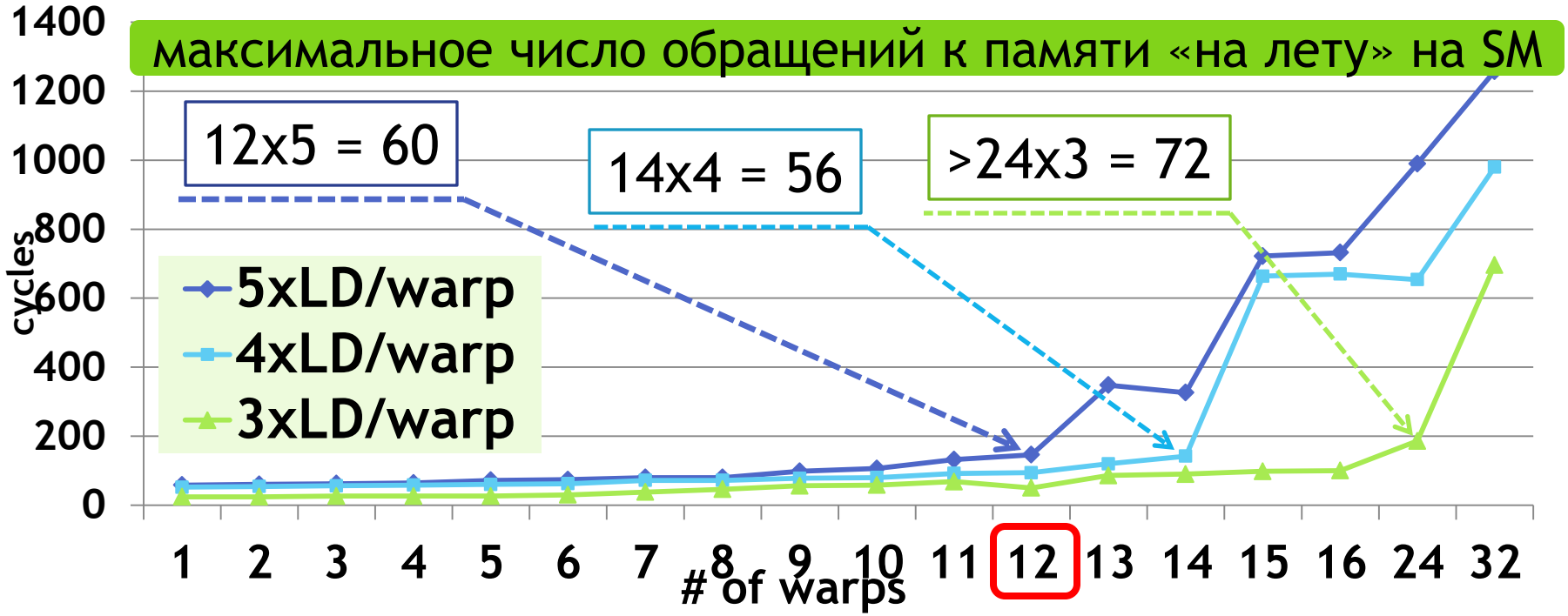
```
LD.E [R8+0x0], R0
LD.E [R8+0x4], R1
LD.E [R8+0x8], R2
LD.E [R8+0xC], R3
LD.E [R8+0x10],R4
IADD R5, R6, R7
```

Instruction fetch – 6 cycles

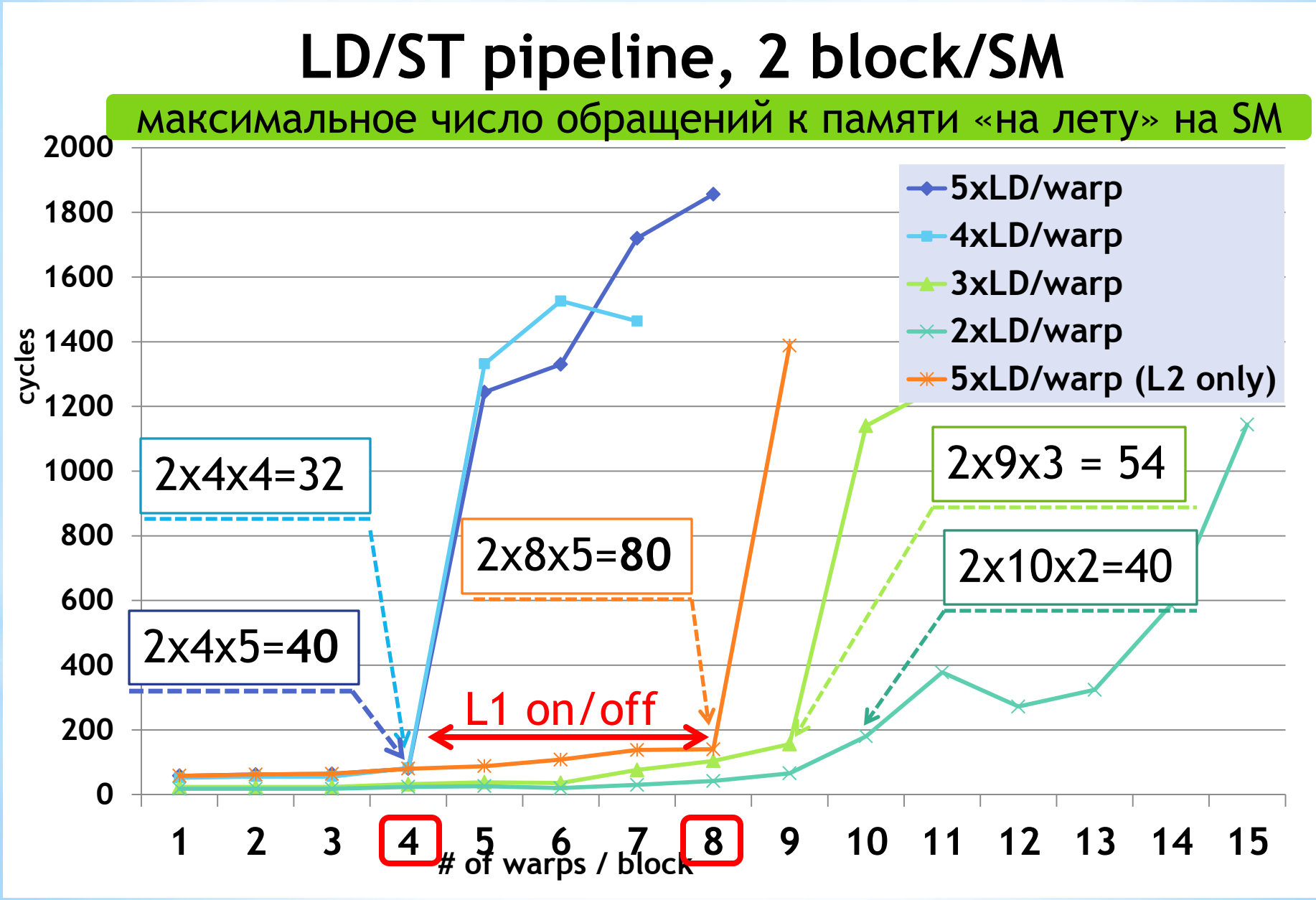
No stall

до 5 подряд идущих незавершенных обращений к памяти на варп!

LD/ST pipeline, 1 block/SM



LD/ST конвейер



LD/ST конвейер

Выводы:

- Необходимо перемешивать инструкции – после 5 LOAD вставлять арифметические, чтобы поток инструкций не блокировался;
- Обращения в память «на лету» поддерживаются вплоть до **12 варпов** (**384 треда**) (1 блок/SM);
- Отключение кеширования в L1 увеличивает число одновременных обращений в память «на лету»;

Таким образом, в GPU реализована поддержка *множественных* незавершенных обращений к памяти из варпа для большого их (варпов) числа

Вычислительный конвейер

- **32 instr/cycle** with **32-bit** operands (most);
- **16 instr/cycle** with **double**;

Min 192 threads (6 warps)/SM

```
IADD R1,R2,R3 //R1=R2+R3
IADD R4,R5,R6 //R4=R5+R6
IADD R7,R4,R6 //R7=R4+R6
```

Instruction fetch – **6** cycles (independent)

Read-after-write dependency – **18-20** cycles

Latency:

IMUL, IADD, FMUL, FADD - **18-20** cycles; DFMA - **22-24** cycles

DADD, DMUL – **22** cycles; FFMA – **18-22** cycles

Необходимо ~20 варпов/SM (640 тредов) для полной загрузки конвейера?

(Для незавершенных обращений в память – **max 320** тредов)

Вычислительный конвейер

- **32 instr/cycle** with **32-bit** operands (most);
- **16 instr/cycle** with **double**;

Min 192 threads (6 warps)/SM

```
IADD R1,R2,R3 //R1=R2+R3
IADD R4,R5,R6 //R4=R5+R6
IADD R7,R4,R6 //R7=R4+R6
```

Instruction fetch - **6** cycles (independent)

Read-after-write dependency - **18-20** cycles

Latency:

IMUL, IADD, FMUL, FADD - **18-20** cycles; DFMA - **22-24** cycles

DADD, DMUL - **22** cycles; FFMA - **18-22** cycles

Thread level parallelism (TLP):

thread 1	thread 2	thread 3
x = x + a	y = y + a	z = z + a
x = x + b	y = y + b	z = z + b
x = x + c	y = y + c	z = z + c

Independent instructions

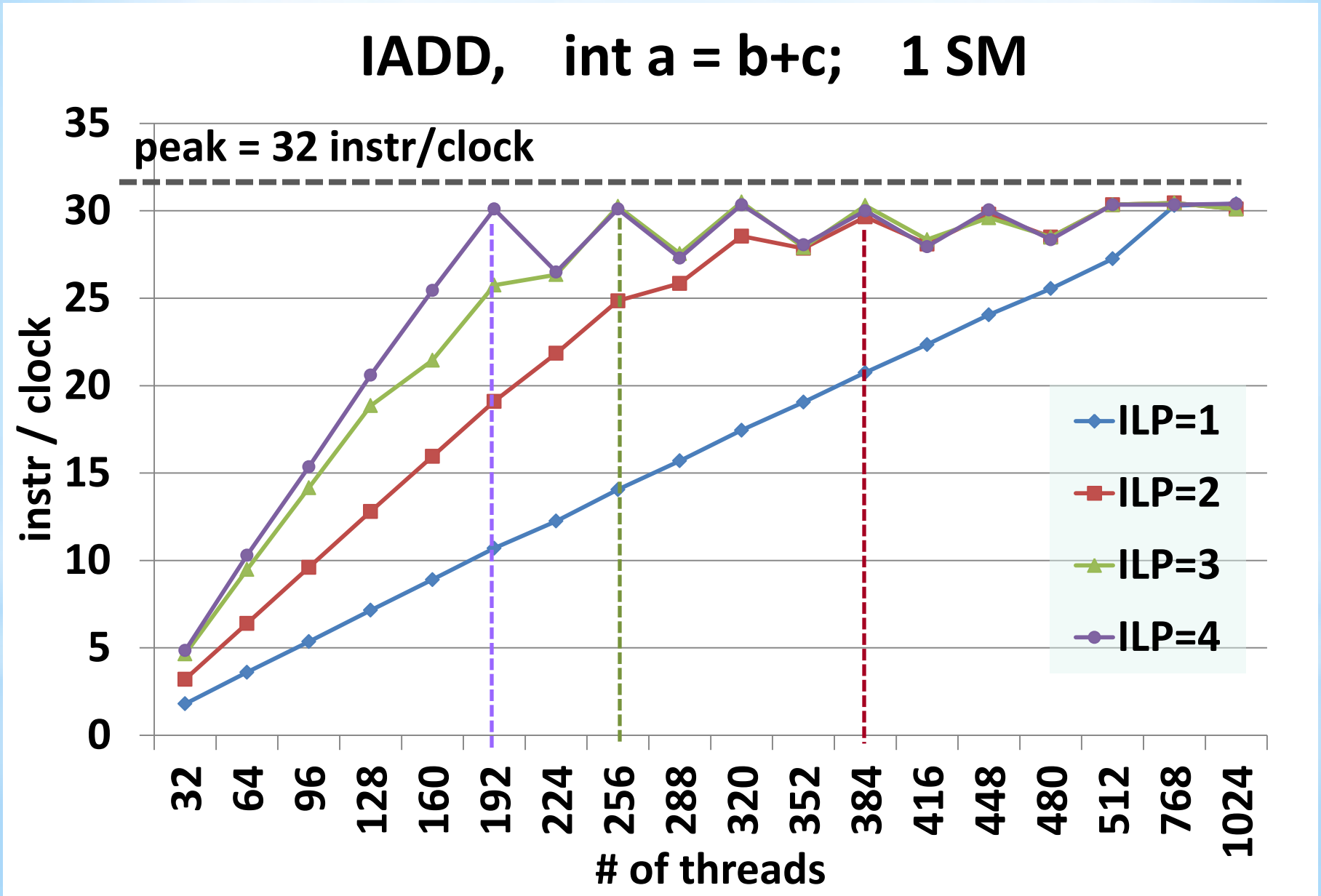
Instruction level parallelism(ILP):

thread
x = x + a
y = y + b
z = z + c

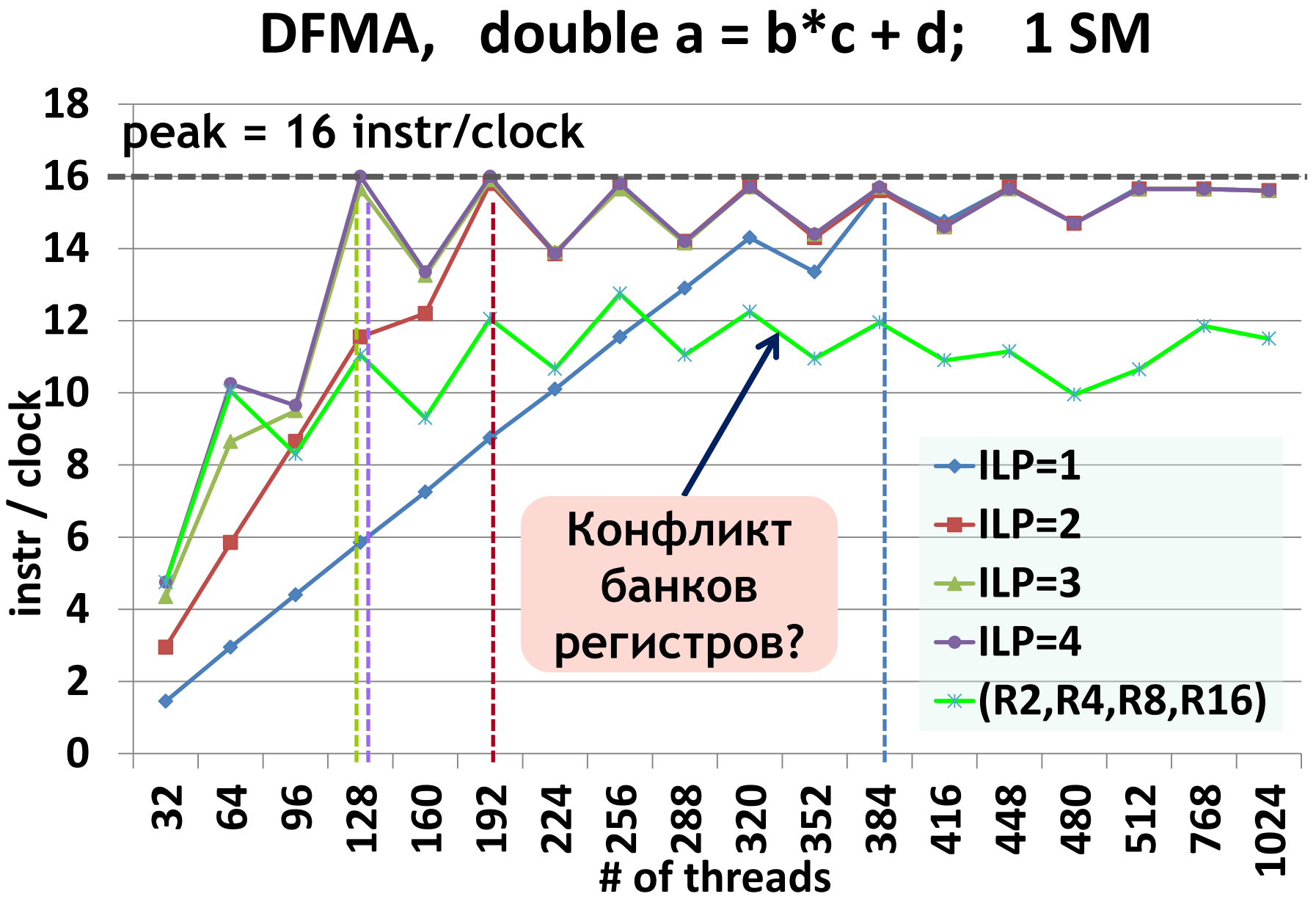
Independent instructions

(ILP = 3)

Вычислительный конвейер



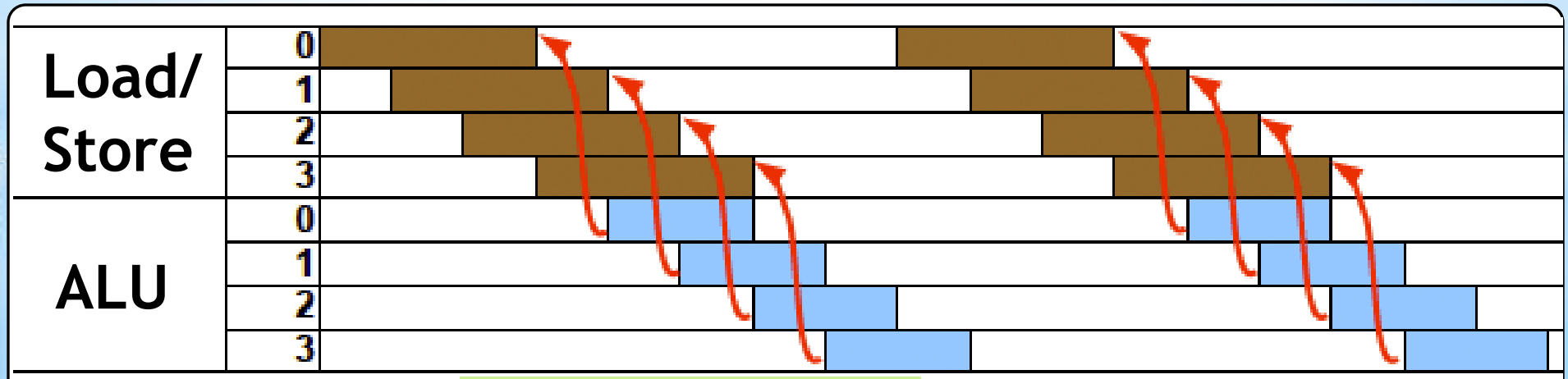
Вычислительный конвейер



TLP & ILP

Thread Level Parallelism (TLP):

1 thread, no Instruction Level Parallelism (ILP):

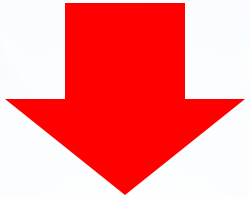


1 thread, with ILP

- зависимость по данным

Выводы

С внутритредовым параллелизмом (ILP=3,4) достаточно лишь **128-192** тредов на SM для полной загрузки выч конвейера



Использование меньшего числа тредов на SM позволяет «держать» незавершенными большее число обращений к памяти на варп, что вместе со знанием латентностей инструкций открывает возможности для тонкой оптимизации кода (prefetch, instruction mix)