PMPP 2015/16

Parallel Basics

(Preliminary) Course Schedule

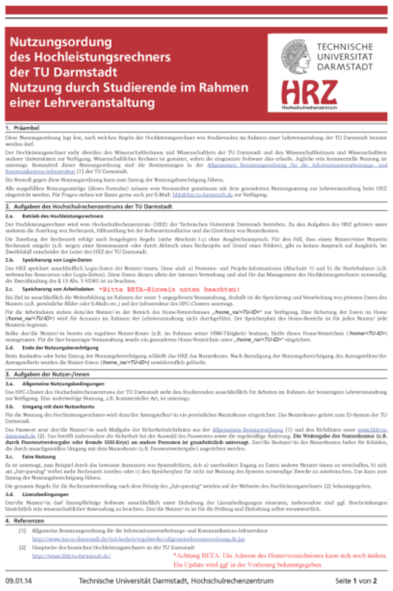
you are here →

12.10.2015	Introduction to PMPP
13.10.2015	Lecture CUDA Programming 1
19.10.2015	Lecture CUDA Programming 2
20.10.2015	Lecture CUDA Programming 3
26.10.2015	Lecture Parallel Basics, Exercise 1 assigned
27.10.2015	Questions and Answers (Q&A), S3|19, Room 2.8
2.11.2015	Introduction Final Projects, Ex. 1 due, Ex. 2 assigned
3.11.2015	Questions and Answers (Q&A)
9.11.2015	Final Projects assigned, Lecture, Exercise 2 due
10.11.2015	Lecture
16.11.2015	Questions and Answers (Q&A)
17.11.2015	Questions and Answers (Q&A)
23.11.2015	1st Status Presentation Final Projects
24.11.2015	1st Status Presentation Final Projects (continued)

HHLR Account for Projects

- please fill in the account form available in Moodle and return to us

This is now overdue!!!



Nutzungsordung des Hochleistungsrechners der TU Darmstadt Nutzung durch Studierende im Rahmen einer Lehrveranstaltung

TECHNISCHE UNIVERSITÄT DARMSTADT

HRZ Hochschulrechenzentrum

1. Präambel

Diese Nutzungsordnung legt fest, nach welchen Regeln der Hochleistungsrechner von Studierenden im Rahmen einer Lehrveranstaltung der TU Darmstadt benutzt werden darf.

Der Hochleistungsrechner steht überdies den Wissenschaftlerinnen und Wissenschaftlern der TU Darmstadt und den Wissenschaftlerinnen und Wissenschaftlern anderer Universitäten zur Verfügung. Wissenschaftliches Rechnen ist gestattet, sofern die eingesetzte Software dies erlaubt. Jegliche rein kommerzielle Nutzung ist untersagt. Bestandteil dieser Nutzungsordnung sind die Bestimmungen in der Allgemeinen Benutzungsordnung für die Informationsverarbeitungs- und Kommunikations-Infrastruktur [1] der TU Darmstadt.

Ein Verstoß gegen diese Nutzungsordnung kann zum Entzug der Nutzungsberechtigung führen.

Alle ausgefüllten Nutzungsanträge (dieses Formular) müssen vom Veranstalter gemeinsam mit dem gesonderten Nutzungsantrag zur Lehrveranstaltung beim HRZ eingereicht werden. Für Fragen stehen wir Ihnen gerne auch per E-Mail: hhlr@hrz.tu-darmstadt.de zur Verfügung.

2. Aufgaben des Hochschulrechenzentrums der TU Darmstadt

2.a. Betrieb des Hochleistungsrechners

Der Hochleistungsrechner wird vom Hochschulrechenzentrum (HRZ) der Technischen Universität Darmstadt betrieben. Zu den Aufgaben des HRZ gehören unter anderem die Zuteilung von Rechenzeit, Hilfestellung bei der Softwareinstallation und das Einrichten von Nutzerkonten.

Die Zuteilung der Rechenzeit erfolgt nach festgelegten Regeln (siehe Abschnitt 3.c) ohne Ausgleichsanspruch. Für den Fall, dass einem Nutzer/einer Nutzerin Rechenzeit entgeht (z.B. wegen einer Systemauszeit oder durch Abbruch eines Rechenjobs auf Grund eines Fehlers), gibt es keinen Anspruch auf Ausgleich. Im Zweifelsfall entscheidet der Leiter des HRZ der TU Darmstadt.

2.b. Speicherung von Login-Daten

Das HRZ speichert ausschließlich Login-Daten der Nutzer/-innen. Diese sind: a) Personen- und Projekt-Informationen (Abschnitt 5) und b) die Betriebsdaten (z.B. verbrauchte Ressourcen oder Login-Zeiten). Diese Daten dienen allein der internen Verwaltung und sind für das Management des Hochleistungsrechners notwendig; die Zweckbindung des § 13 Abs. 5 HDSG ist zu beachten.

2.c. Speicherung von Arbeitsdaten *Bitte BETA-Hinweis unten beachten!

Ein Ziel ist ausschließlich die Weiterbildung im Rahmen der unter 5 angegebenen Veranstaltung, deshalb ist die Speicherung und Verarbeitung von privaten Daten des Nutzers (z.B. persönliche Bilder oder E-Mails etc.) auf der Infrastruktur des HPC-Clusters untersagt.

Für die Arbeitsdaten stehen dem/der Nutzer/-in der Bereich des Home-Verzeichnisses „/home_na/<TU-ID>" zur Verfügung. Eine Sicherung der Daten im Home (/home_na/<TU-ID>) wird für Accounts im Rahmen der Lehrveranstaltung nicht durchgeführt. Der Speicherplatz des Home-Bereichs ist für jeden Nutzer/ jede Nutzerin begrenzt.

Sollte der/die Nutzer/-in bereits ein reguläres Nutzer-Konto (z.B. im Rahmen seiner HiWi-Tätigkeit) besitzen, bleibt dieses Home-Verzeichnis (/home/<TU-ID>) unangetastet. Für die hier beantragte Veranstaltung wurde ein gesondertes Home-Verzeichnis unter „/home_na/<TU-ID>" eingerichtet.

2.d. Ende der Nutzungsberechtigung

Beim Auslaufen oder beim Entzug der Nutzungsberechtigung schließt das HRZ das Nutzerkonto. Nach Beendigung der Nutzungsberechtigung des Antragstellers/der Antragstellerin werden die Nutzer-Daten (/home_na/<TU-ID>) unwiderruflich gelöscht.

3. Aufgaben der Nutzer-/innen

3.a. Allgemeine Nutzungsbedingungen

Das HPC-Cluster des Hochschulrechenzentrums der TU Darmstadt steht den Studierenden ausschließlich für Arbeiten im Rahmen der beantragten Lehrveranstaltung zur Verfügung. Eine anderweitige Nutzung, z.B. kommerzieller Art, ist untersagt.

3.b. Umgang mit dem Nutzerkonto

Für die Nutzung des Hochleistungsrechners wird dem/der Antragsteller/-in ein persönliches Nutzerkonto eingerichtet. Das Nutzerkonto gehört zum ID-System der TU Darmstadt.

Das Passwort setzt der/die Nutzer/-in nach Maßgabe der Sicherheitsrichtlinien aus der Allgemeinen Benutzerordnung [1] und den Richtlinien unter www.hhlr.tu-darmstadt.de [2]. Das betrifft insbesondere die Sicherheit bei der Auswahl des Passwortes sowie die regelmäßige Änderung. **Die Weitergabe des Nutzerkontos (z.B. durch Passwortweitergabe oder fremde SSH-Keys) an andere Personen ist grundsätzlich untersagt.** Der/die Besitzer/-in des Nutzerkontos haftet für Schäden, die durch unsachgemäßen Umgang mit dem Nutzerkonto (z.B. Passwortweitergabe) angerichtet werden.

3.c. Faire Nutzung

Es ist untersagt, zum Beispiel durch das bewusste Ausnutzen von Systemfehlern, sich a) unerlaubten Zugang zu Daten anderer Nutzer/-innen zu verschaffen, b) sich am „fair-queuing" vorbei mehr Rechenzeit zuteilen oder c) den Speicherplatz für nicht zur Nutzung des Systems notwendige Zwecke zu missbrauchen. Das kann zum Entzug der Nutzungsberechtigung führen.

Die genauen Regeln für die Rechenzeitverteilung nach dem Prinzip des „fair-queuing" werden auf der Webseite des Hochleistungsrechners [2] bekanntgegeben.

3.d. Lizenzbedingungen

Der/die Nutzer/-in darf lizenzpflichtige Software ausschließlich unter Einhaltung der Lizenzbedingungen einsetzen, insbesondere sind ggf. Beschränkungen hinsichtlich rein wissenschaftlicher Anwendung zu beachten. Der/die Nutzer/-in ist für die Prüfung und Einhaltung selbst verantwortlich.

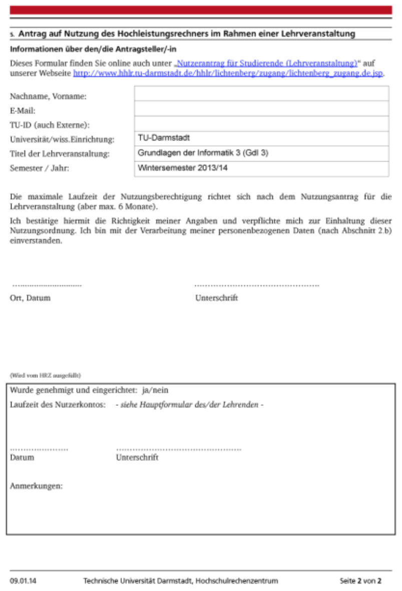
4. Referenzen

[1] Allgemeine Benutzungsordnung für die Informationsverarbeitungs- und Kommunikations-Infrastruktur http://www.hrz.tu-darmstadt.de/itsicherheit/regelwerke/allgemeinebenutzerordnung.de.jsp

[2] Hauptseite des hessischen Hochleistungsrechners an der TU Darmstadt http://www.hhlr.tu-darmstadt.de/

*Achtung BETA: Die Adresse des Homeverzeichnisses kann sich noch ändern. Ein Update wird ggf in der Vorlesung bekanntgegeben.

09.01.14 Technische Universität Darmstadt, Hochschulrechenzentrum Seite 1 von 2



5. Antrag auf Nutzung des Hochleistungsrechners im Rahmen einer Lehrveranstaltung

Informationen über den/die Antragsteller/-in

Dieses Formular finden Sie online auch unter „Nutzerantrag für Studierende (Lehrveranstaltung)" auf unserer Webseite http://www.hhlr.tu-darmstadt.de/hhlr/lichtenberg/zugang/lichtenberg_zugang.de.jsp.

Nachname, Vorname:	
E-Mail:	
TU-ID (auch Externe):	
Universität/wiss.Einrichtung:	TU-Darmstadt
Titel der Lehrveranstaltung:	Grundlagen der Informatik 3 (GdI 3)
Semester / Jahr:	Wintersemester 2013/14

Die maximale Laufzeit der Nutzungsberechtigung richtet sich nach dem Nutzungsantrag für die Lehrveranstaltung (aber max. 6 Monate).

Ich bestätige hiermit die Richtigkeit meiner Angaben und verpflichte mich zur Einhaltung dieser Nutzungsordnung. Ich bin mit der Verarbeitung meiner personenbezogenen Daten (nach Abschnitt 2.b) einverstanden.

....

Ort, Datum Unterschrift

(Wird vom HRZ ausgefüllt)

Wurde genehmigt und eingerichtet: ja/nein

Laufzeit des Nutzerkontos: *- siehe Hauptformular des/der Lehrenden -*

....

Datum Unterschrift

Anmerkungen:

09.01.14 Technische Universität Darmstadt, Hochschulrechenzentrum Seite 2 von 2

Regarding your HHLR accounts

- If you only gave us your matriculation number **but not your TU-ID** on the HHLR account registration forms **you will not yet have a working account**

- In that case **send us an email today** to pmpp2015@gris.informatik.tu-darmstadt.de with:
  - Your **matriculation number** (that you already provided on the form)
  - Your **TU-ID**

- This will allow us to associate a TU-ID with you
  - The TU-ID is your username for the HHLR

!

Exercise 1 – Matrix Multiplication

- Your task: implement a simple matrix multiplication as shown in the lecture
  - CPU and GPU
  - A naïve version (w/o shared memory usage) is sufficient
  - But must be able to use more than one block

- Purpose of this task is to get you acquainted with CUDA and working on the HHLR

- We provide you with a code skeleton that you will fill in
  - You can change it any way you want
  - But it must compile and run on the HHLR using only **`cmake`**, **`make`** and **`bsub`**

Exercises and Final Projects

- Important remark: If you are unable to complete Exercises 1 and 2, you will (most likely) also fail the projects!
- Urgent advice: COMPLETE THE EXERCISES!
- Urgent advice (2): I REALLY MEAN IT …

!

Fundamentals of Parallel Computing

- parallel computing requires that
  - the problem can be decomposed into sub-problems that can be safely solved at the same time
  - the programmer structures the code and data to solve these sub-problems concurrently

- the goals of parallel computing are
  - to solve problems in less time, and/or
  - to solve bigger problems

➔ problems must be large enough to justify parallel computing and to exhibit exploitable concurrency

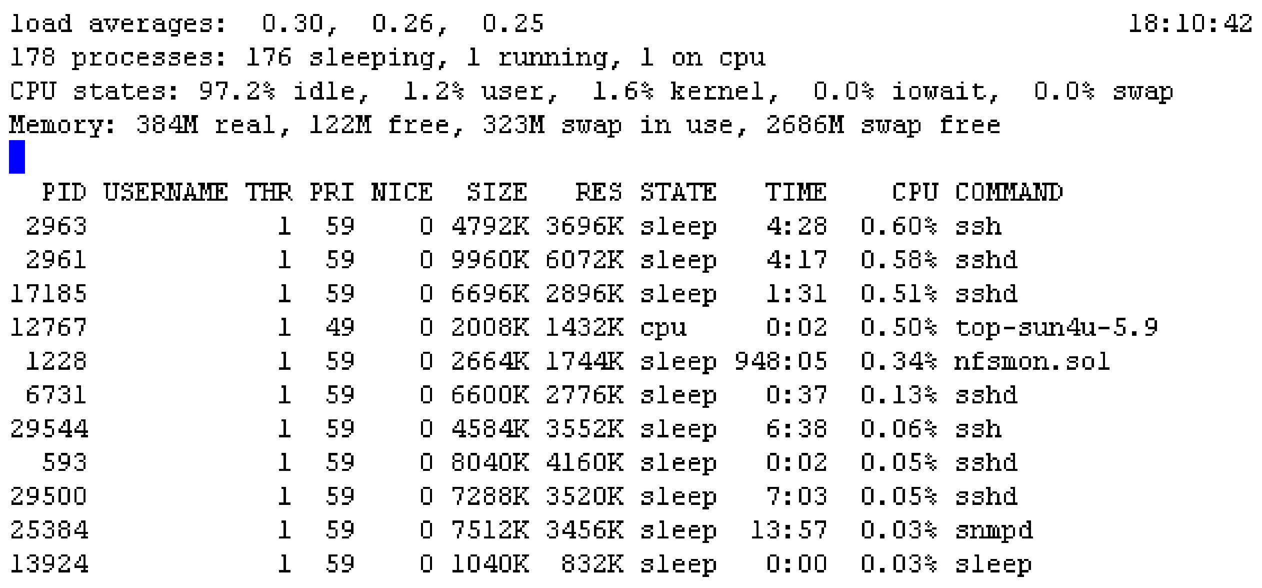
Recommended Reading

- Preface, Chapters 1+2 of Patterns for Parallel Programming by Mattson, Sanders, and Massingill
  - design patterns for parallel programming
  - background and jargon

- lecture slides follow their approach

(UNIX) Operating System

- parallelism is obvious
- multiple users, processes, pipes (consumer/producer), …
- processes may not interfere
- goal is related to throughput, response time



```
load averages:  0.30,  0.26,  0.25                                   18:10:42
178 processes: 176 sleeping, 1 running, 1 on cpu
CPU states: 97.2% idle,  1.2% user,  1.6% kernel,  0.0% iowait,  0.0% swap
Memory: 384M real, 122M free, 323M swap in use, 2686M swap free

  PID USERNAME THR PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND
 2963            1  59    0 4792K 3696K sleep   4:28  0.60% ssh
 2961            1  59    0 9960K 6072K sleep   4:17  0.58% sshd
17185            1  59    0 6696K 2896K sleep   1:31  0.51% sshd
12767            1  49    0 2008K 1432K cpu     0:02  0.50% top-sun4u-5.9
 1228            1  59    0 2664K 1744K sleep 948:05  0.34% nfsmon.sol
 6731            1  59    0 6600K 2776K sleep   0:37  0.13% sshd
29544            1  59    0 4584K 3552K sleep   6:38  0.06% ssh
  593            1  59    0 8040K 4160K sleep   0:02  0.05% sshd
29500            1  59    0 7288K 3520K sleep   7:03  0.05% sshd
25384            1  59    0 7512K 3456K sleep  13:57  0.03% snmpd
13924            1  59    0 1040K  832K sleep   0:00  0.03% sleep
```

Examples of Parallel Systems

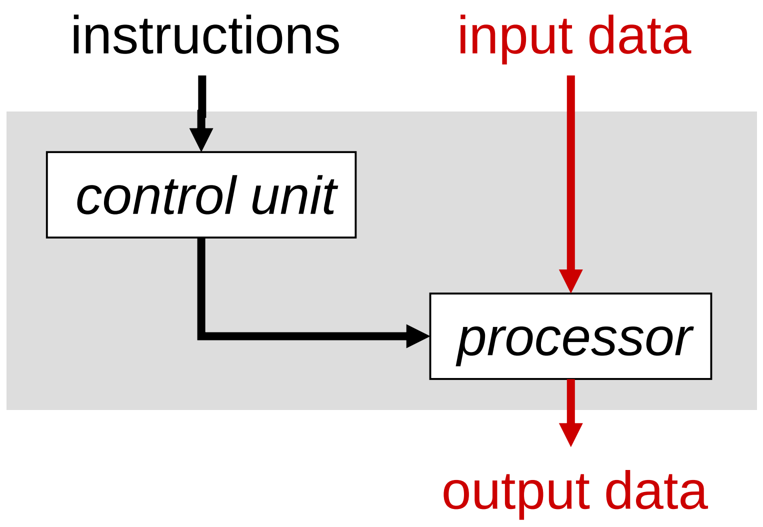
A Parallel Program

- finding and exploiting parallelism can be a challenge
  - even more problematic given hardware constraints
- overall goal is minimizing total running time of a single program

Brief Intro to Parallel Architectures

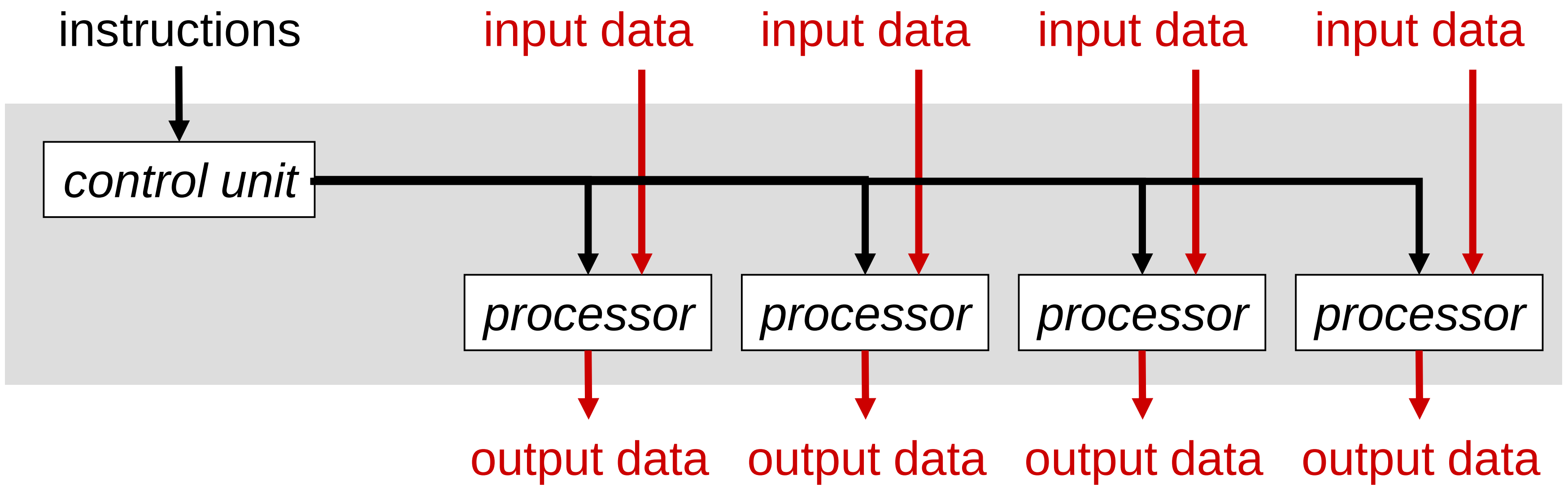
- Flynn's Taxonomy [Flynn 1972]

- Single Instruction, Single Data (SISD)
  - common von Neumann model
  - used by virtually all single-processor computers



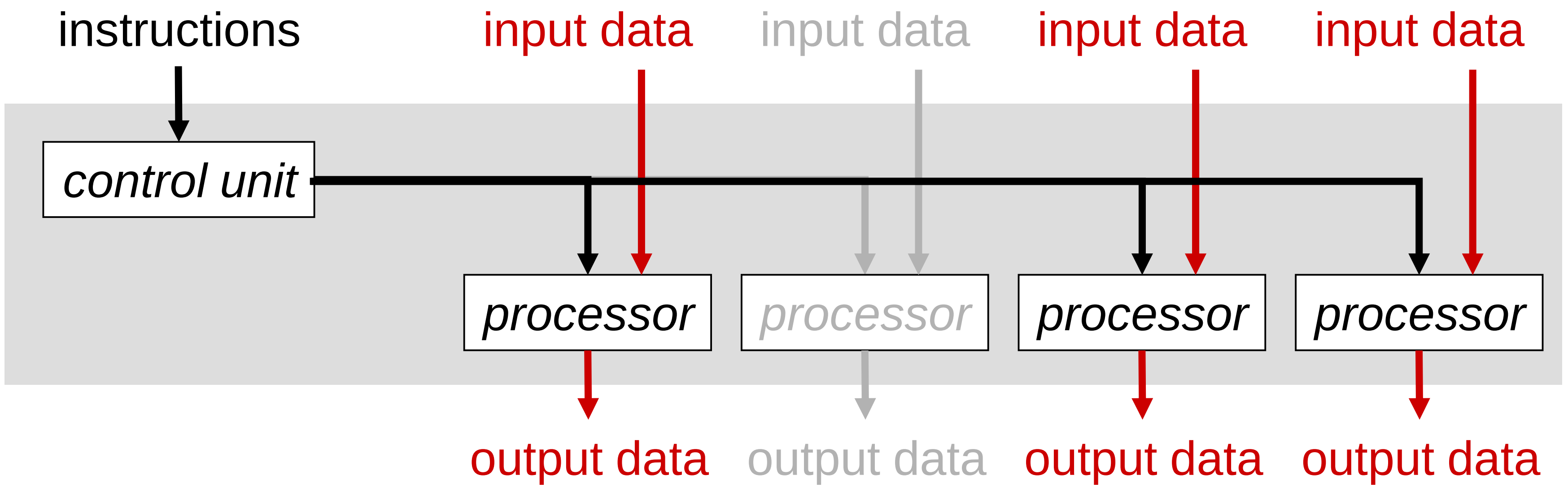
Brief Intro to Parallel Architectures

- Single Instruction, Multiple Data (SIMD)
  - single instruction stream broadcast to multiple processors
  - separate data stream per processor
  - typically fine-grained parallelism, little interprocess communication
  - also vector processors (parallelism usually exploited by compiler)



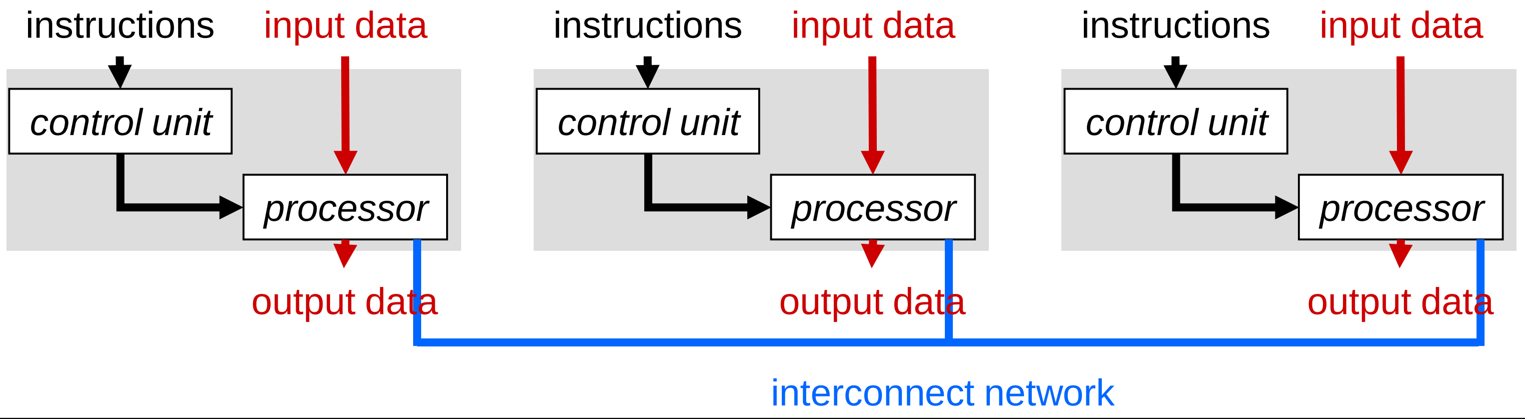
Brief Intro to Parallel Architectures

- Single Instruction, Multiple Thread (SIMT)
  - not part of Flynn's original taxonomy
  - single instruction stream broadcast to multiple processors
  - instruction may or may not be executed by a processor depending on some conditions
  - CUDA warps follow exactly this model



Brief Intro to Parallel Architectures

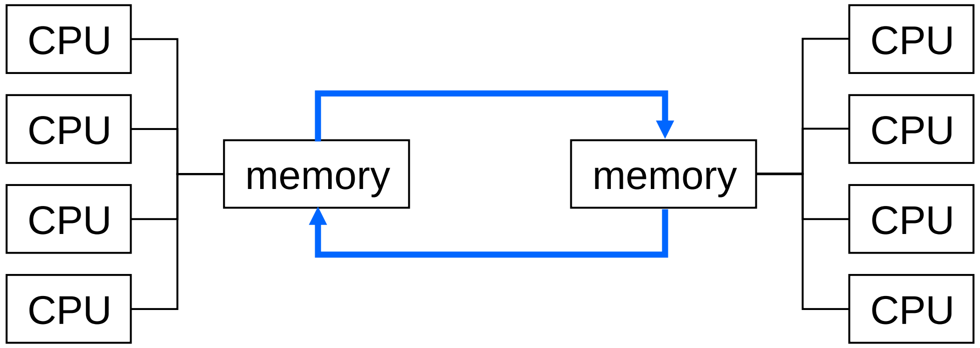
- Multiple Instruction, Single Data (MISD)
  - no well known systems
- Multiple Instruction, Multiple Data (MIMD)
  - instruction and data stream per processor
  - most general model
  - typical model for cluster
  - CUDA blocks approximate this model



MIMD Breakdown: Shared Memory

- symmetric multiprocessors (SMP)
  - all processors share connection to same memory
  - bandwidth typically limiting factor
  - does not scale well, only small number of processors

- non-uniform memory access (NUMA)
  - memory uniformly addressable from all processors
  - access speed depends significantly on location in memory
  - can be alleviated using caches (cache coherent NUMA)
  - logically same programming as SMP but performance issues



MIMD Breakdown: Distributed Memory

- separate address space per process
- communication using message passing (send/receive messages)
- all communication must be explicitly programmed

- massively parallel processors (MPP)
  - processors and network infrastructure tightly coupled
  - very scalable, supporting thousands of processors in a system

- clusters
  - composed of off-the-shelf components
  - e.g., Linux Beowulf cluster

MIMD Breakdown: Hybrid Systems

- clusters of nodes with separate address spaces
- several processors per node with shared memory
- fast communication

MIMD Breakdown: Grids

- systems that use distributed heterogeneous resources connected via WAN/LAN
- interconnection also via Internet
- example: Linux cluster in GRIS
  - use idle cycles of PC in terminal room for projects
  - should take into account whether PC is currently used locally

Breakdown Summary

- systems classified according to hardware architecture
- influences typically the native parallel programming model
- can be hidden by middleware
    - e.g., virtual distributed shared memory systems

Parallel Programming Jargon

- definitions of frequently used terms

- task
  - sequence of instructions that operate as a group
  - first step is typically breaking the problem down into tasks
  - possible task breakdowns for matrix multiplication:
    - task = multiplication of sub-blocks of the matrix
      (see the CUDA example)
    - task = dot product between a row and a column vector
    - task = individual iterations of the loops used for matrix multiplication

  ➔ definition of task depends on the way the programmer thinks about the problem

Parallel Programming Jargon

- unit of execution (UE)
  - a task is mapped for execution on a UE
  - UE can be process or thread
  - process
    - a collection of resources enabling the execution of program instructions
    - e.g., virtual memory, I/O descriptors, stack
    - “heavyweight” execution unit with own address space
  - thread
    - fundamental UE in modern operating systems
    - shares the environment of the process with other threads

    ➔ lightweight, fast context switch
  - CUDA thread
    - “ultra lightweight” threads

Parallel Programming Jargon

- processing element (PE)
  - generic term for hardware unit executing a stream of instructions
  - could be processor, workstation, …

- load balance, load balancing
  - tasks must be mapped to UEs
  - UEs must be mapped to PEs
  - mapping is crucial for overall performance
    - e.g., bad if most PEs are idle while a small number is performing some computation
  - load balance
    - how well is work balanced between PEs
  - load balancing
    - process of static or dynamic allocation of work to PEs
    - goal: even distribution

Parallel Programming Jargon

- synchronous vs. asynchronous
  - events that must happen at the same time are synchronous
  - otherwise they are asynchronous

- synchronization
  - order of execution in general nondeterministic
  - synchronization enforces a particular order of events (ordering constraint) if necessary
  - CUDA primitives
    - syncthreads, atomic operations, …

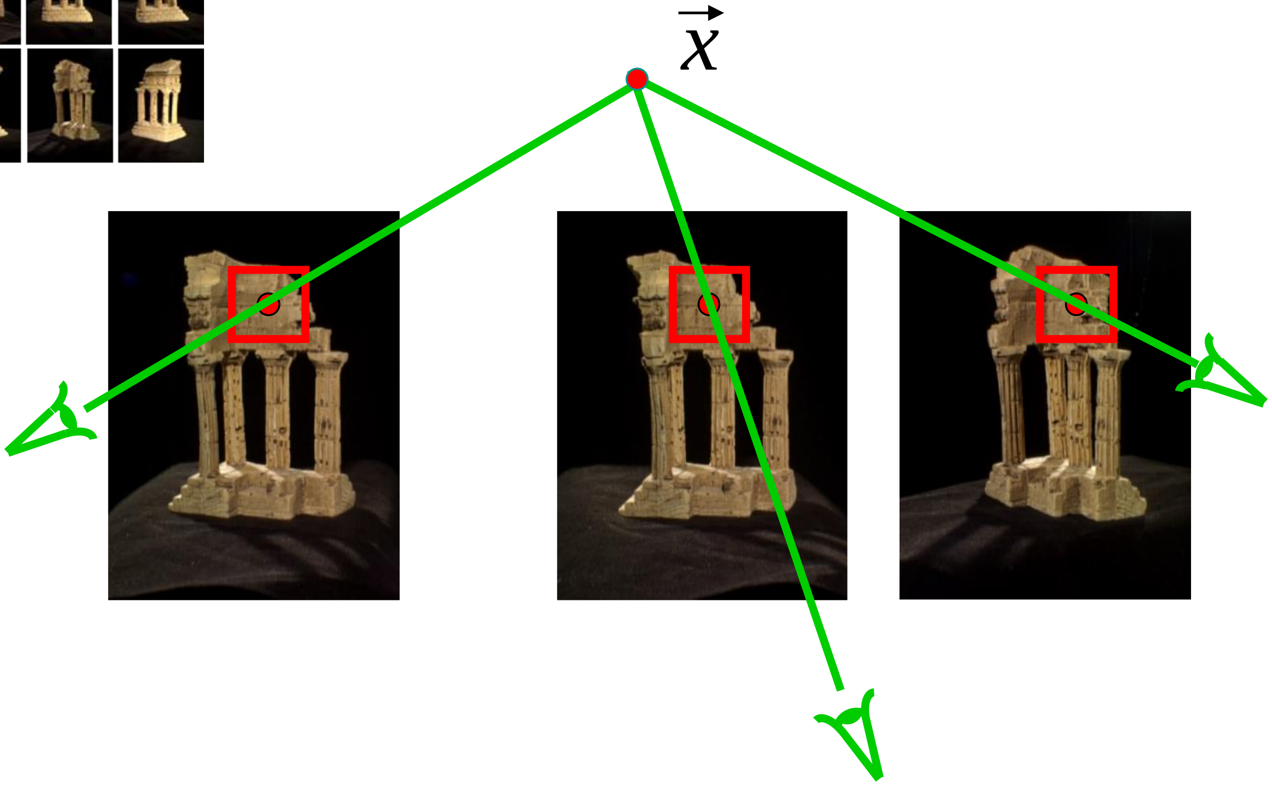
CUDA Concept: Atomic Operations

- performs a read-modify-write atomic operation on one 32-bit or 64-bit word residing in global or shared memory
- guaranteed to be performed without interference from other threads
- example: **`atomicAdd()`**
- not available on all devices, see details in Appendix C

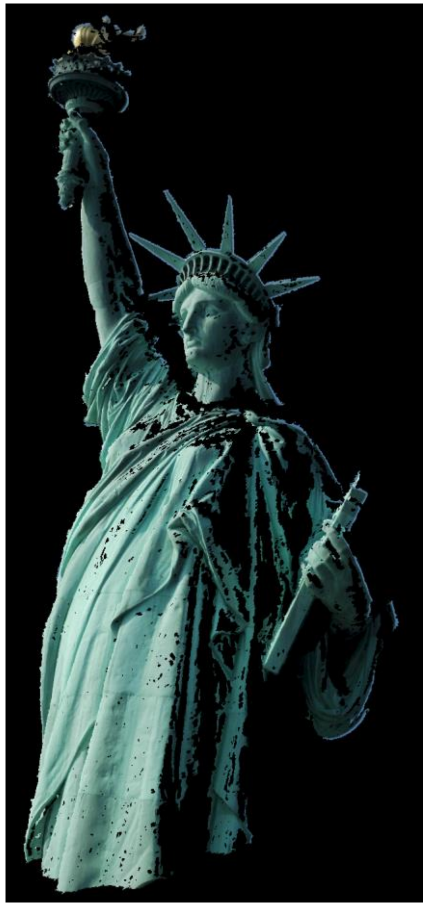
Parallel Programming Jargon

- race condition
  - error (effect) peculiar to parallel programs
  - outcome of the program changes as the relative order (scheduling) of UEs changes
  - same program run twice on the identical system might yield different results
  - debugging hard as race conditions cannot be reliably reproduced
  - some tools available for detection

Race Condition Example

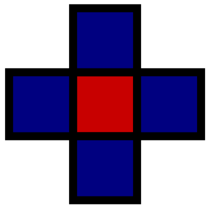


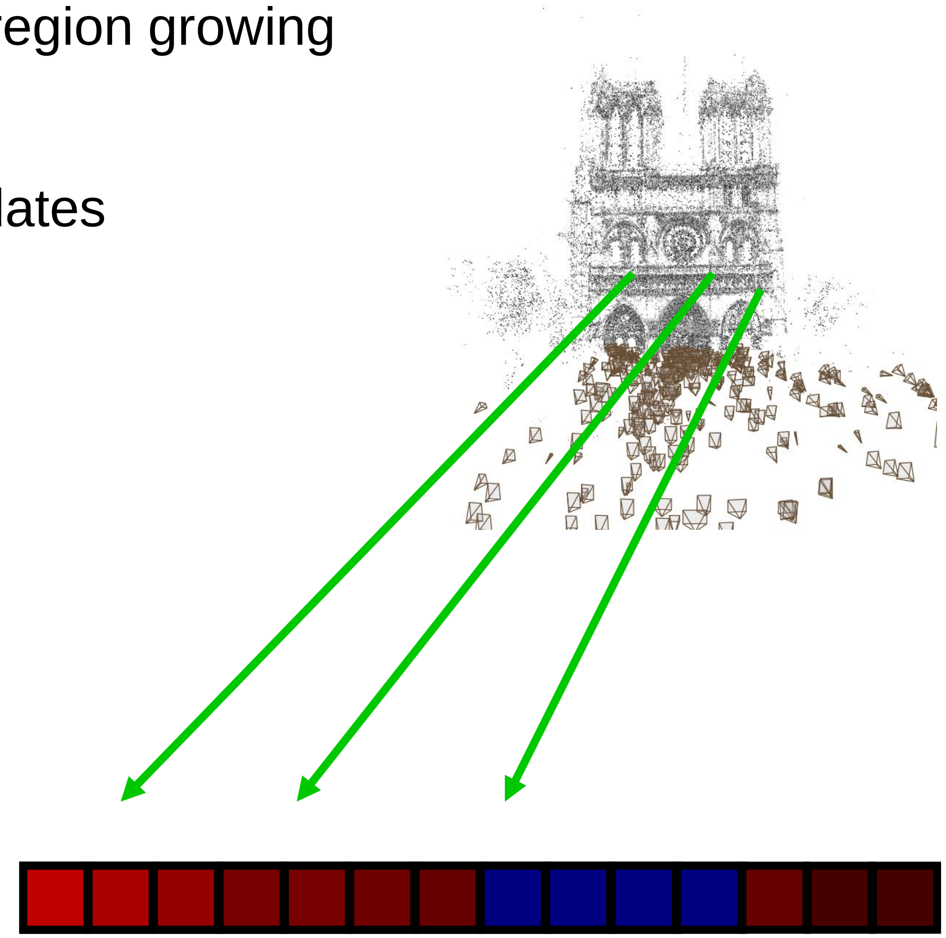
Race Condition Example



Race Condition Example

- race condition example: parallel region growing
- priority queue of matching candidates
  - ordered by matching confidence
  - initialization by SfM features
- process front of queue
  - on success
    return neighbors to queue





Parallel Programming Jargon

- deadlocks
  - error peculiar to parallel programs
  - cycle of tasks where each task is blocked waiting for another block

  ➔ blocked forever

  - easy to discover as tasks stop at this point

  - CUDA
    - syncthreads basically deadlocks when not all threads reach the syncthread statement
    - handled (but not corrected!) by hardware

Quantifying Parallel Computation

- modeling and measuring the performance of a parallel system

- execution time of a program run on a single PE

$$T_{\text{total}}(1) = T_{\text{setup}} + T_{\text{compute}} + T_{\text{finalization}}$$

- execution on a parallel computer
  - use P PEs to compute

$$T_{total}(P) = T_{setup} + \frac{T_{compute}(1)}{P} + T_{finalization}$$

  - very idealized situation but shows important idea of serial and parallelizable parts in computation

Quantifying Parallel Computation

- relative speedup S
  - how much faster is a program running
  - removes actual running time of the program

$$S(P) = \frac{T_{total}(1)}{T_{total}(P)}$$

- efficiency E
  - speedup normalized by the number of PEs

$$E(P) = \frac{S(P)}{P} = \frac{T_{total}(1)}{P \cdot T_{total}(P)}$$

- perfect linear speedup if $E(P) = 1$
  - can rarely be achieved

Quantifying Parallel Computation

- serial fraction γ
  - relative running time of serial terms that cannot be parallelized

$$\gamma = \frac{T_{setup} + T_{finalization}}{T_{total}(1)}$$

  - fraction of time spent in parallelizable part is (1 - γ)
  - rewriting total computation time using P PEs

$$T_{total}(P) = \gamma \cdot T_{total}(1) + \frac{(1-\gamma) \cdot T_{total}(1)}{P}$$

Quantifying Parallel Computation

- Amdahl's law
  - relative speedup of an ideal parallel algorithm, no overhead in parallel part

$$S(P) = \frac{T_{total}(1)}{\left(\gamma + \frac{1-\gamma}{P}\right) \cdot T_{total}(1)} = \frac{1}{\gamma + \frac{1-\gamma}{P}}$$

  - limit for $P \rightarrow$ infinity

$$S = \frac{1}{\gamma}$$

  - upper bound for speedup obtainable if serial part represents γ of the total computation

Quantifying Parallel Computation

- Amdahl's law in practice
  - how much effort should be spent on parallel implementation if even 10% of the algorithm is serial?
- CUDA: initial setup time for a kernel
- CUDA: transfer time host ↔ device can be a serious bottleneck
- Amdahl's law may be too pessimistic
  - what resources limit computation (memory?)
  - assumes that parallel and serial computation perform identically
  - looks at fixed-size speedup

Quantifying Parallel Computation

$$T_{total}(1) = T_{setup} + P \cdot T_{compute}(P) + T_{finalization}$$

- introducing scaled serial fraction γ_{scaled}

$$\gamma_{scaled} = \frac{T_{setup} + T_{finalization}}{T_{total}(P)}$$

$$T_{total}(1) = \gamma_{scaled} \cdot T_{total}(P) + P \cdot (1 - \gamma_{scaled}) \cdot T_{total}(P)$$

- Gustafson's law: scaled (or fixed-time) speedup

$$S(P) = P + (1 - P) \cdot \gamma_{scaled}$$

  - adding more processors while increasing the problem size can yield linear speedup for constant γ_{scaled}

Communication

- time for message transfer

$$T_{message\_transfer} = \alpha + \frac{N}{\beta}$$

- latency α (unit: time)
  - time it takes to send an empty message
  - overhead due to software and network hardware plus time it takes to traverse the communication medium
- bandwidth β (unit: bytes per time)
  - capacity of the communication medium
  - N is message length

Communication

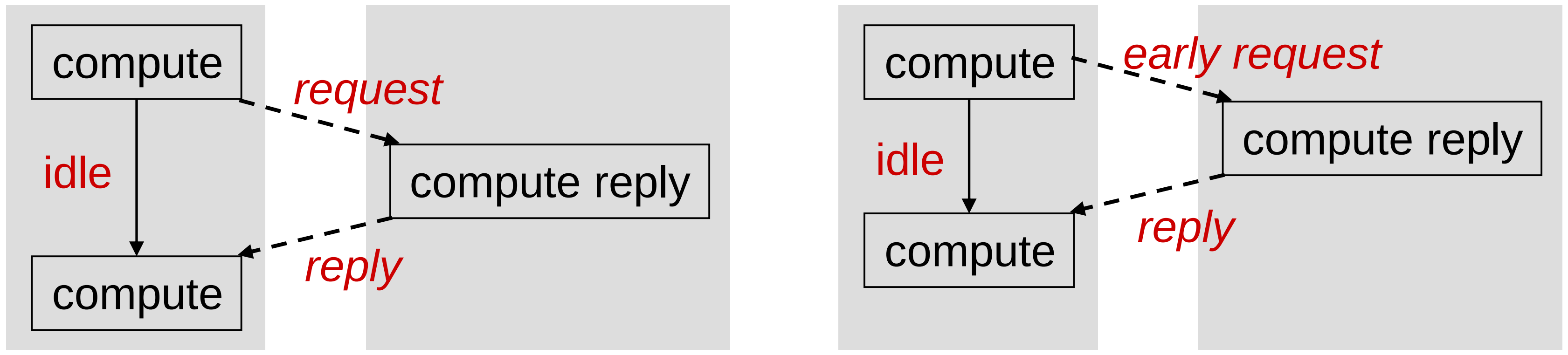
- CUDA: memory transfer host ↔ device, device ↔ device
  - see bandwidth test program in SDK

```
Host to Device Bandwidth for Pageable memory
Transfer Size (Bytes)   Bandwidth(MB/s)
2098.0                  33554432
[…]
```

- CUDA: also relevant when accessing global memory

Communication

- overlapping communication and computation, latency hiding
  - computation while data is sent or received
  - send request for data early
  - ➔ reduces idle time



- assign multiple UE to each PE
  - context switch while a UE waits for communication
  - keeps PE busy

Communication

- CUDA: streams concept
  - compute capability 1.1 and higher allows to overlap memory transfer host ↔ device with computation on device
  - streams of memory transfer and computation
  - see simpleStreams test program in SDK

memcopy:	17.06	
kernel:	41.15	
non-streamed:	55.17	(58.20 expected)
8 streams:	43.30	(43.28 expected with compute capability 1.1 or later)

Communication

- CUDA: latency hiding when accessing global memory
  - using multiple warps per multiprocessor
  - cannot hide bandwidth limitations
  - (not counting any cache)