Kuliah Umum Matematika:

Pengenalan Komputasi Berbasis GPU

7 Oktober 2016 – FMIPA, Universitas Andalas

Muhammad Teguh Satria, MSc

NovaGlobal Pte Ltd
Green & Scientific Computing Solutions

NVIDIA PREFERRED SOLUTION PROVIDER

Profil

Muhammad Teguh Satria

Application Systems Manager – NovaGlobal Pte Ltd, Singapore

Email: mteguhsat@gmail.com , teguh@novaglobal.com.sg

Pendidikan:

- S2, Computer Science, National Taipei University of Technology (Taipei Tech), Taiwan.
- S1, Matematika, Institut Teknologi Bandung, Indonesia.

Pengalaman Penelitian:

- *GPU Performance Model* – Advanced Digital Sciences Center, Singapore.
- Implementasi GPU untuk:
  - *Attentive Teleconferencing Robot* – Advanced Digital Sciences Center, Singapore.
  - *Perkiraan Cuaca (WRF)* – SSEC, University of Wisconsin-Madison, US.
  - *Tsunami Model* – SSEC, University of Wisconsin-Madison, US.
  - *Geothermal* – TerraMath Indonesia
  - *Tsunami Simulation* – Taipei Tech, Taiwan.

Pengenalan Komputasi Berbasis GPU

- Konsep Komputasi Paralel
- Konsep Komputasi Berbasis GPU
- Metode Komputasi Berbasis GPU
- Contoh Kasus

- Konsep Komputasi Paralel
- Konsep Komputasi Berbasis GPU
- Metode Komputasi Berbasis GPU
- Contoh Kasus

Komputasi Paralel

Teknik mempersingkat komputasi dengan cara mengeksekusi instruksi-instruksi di dalamnya secara simultan.

Proses Komputasi di *Operating System*

- Vector addition

  $a + b = c$

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix}$$

```
N = 8;
for( int i = 0; i < N; i++ )
{
        c[i] = a[i] + b[i];
}
```

Aplikasi: vector_addition

process_0
start

thread_0:
- *ambil elemen a_0 di data a*
- *ambil elemen b_0 di data b*
- *tambahkan a_0 dan b_0*
- *simpan hasilnya sebagai elemen c_0 di data c*
- *ambil elemen a_1 di data a*
- *ambil elemen b_1 di data b*
- *tambahkan a_1 dan b_1*
- *…*
- *…*
- *…*
- *exit*

end

Definisi Dalam *Operating System*

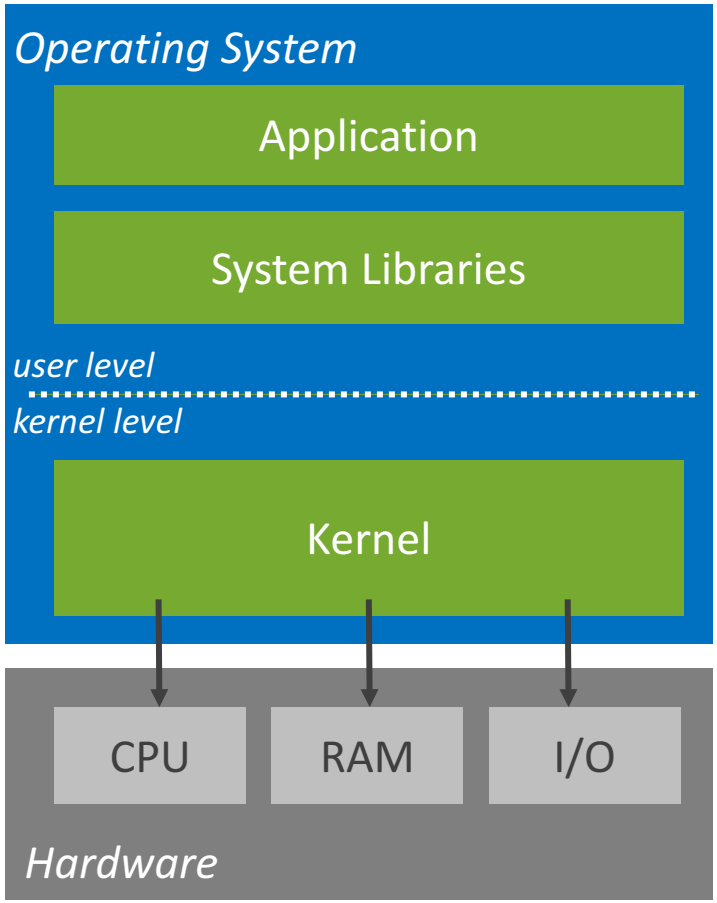
- *process*
  - = operasi yang dijalankan oleh *operating system*
  - memiliki *virtual address space*
  - memiliki unique id
  - sedikitnya memiliki sebuah *thread*

- *thread*
  - = kumpulan perintah yang dieksekusi di processor
  - memiliki unique id
  - bagian dari *process*
  - *thread-thread* dalam sebuah process berbagi *virtual address space*

- Reference
  - *https://msdn.microsoft.com/en-us/library/windows/desktop/ms681917(v=vs.85).aspx*

Thread

- = kumpulan perintah yang dieksekusi di processor
- thread dibedakan dalam 2 tipe: *user thread* and *kernel thread*
- *user thread*
  - User level
  - Manageable
  - Handled by system library
  - Unlimited (depends on memory)
- *kernel thread*
  - kernel level
  - Unmanageable
  - Handled by kernel
  - Depends on number of cores or features in processor (hyperthreading)



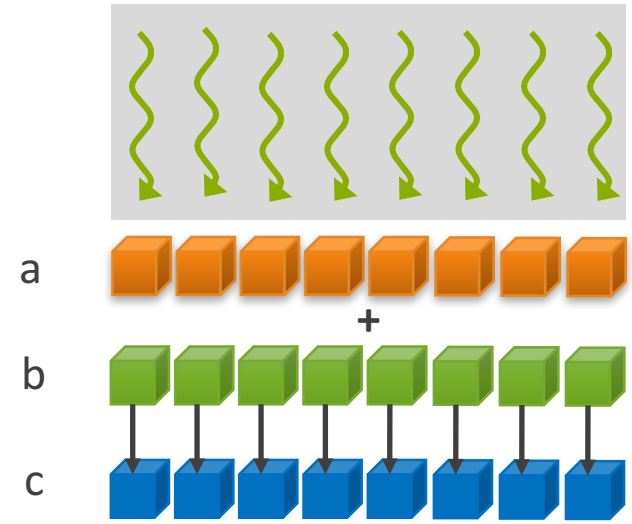
Komputasi Paralel

- Vector addition

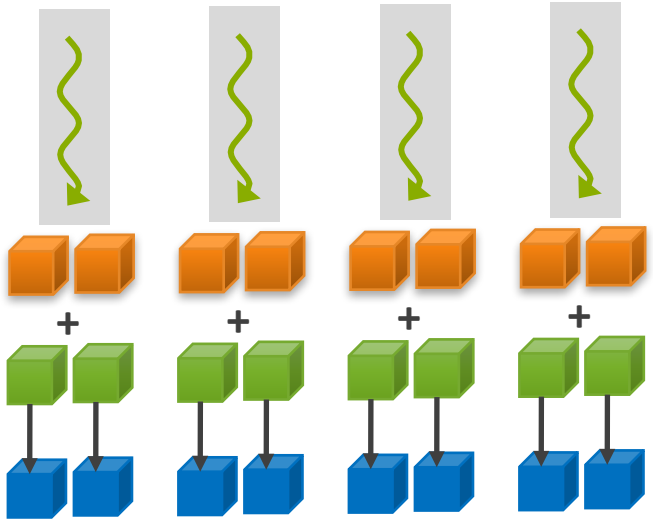
  $a + b = c$

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix}$$

- *Multithreading*



- *Multiprocessing*



Komputasi Parallel - Multithreading

process_0: **vector addition**
start

thread_0: create 7 more threads

thread_0:	*thread_1:*	*thread_2:*	*thread_3:*	*thread_4:*	*thread_5:*	*thread_6:*	*thread_7:*
- ambil elemen a_0 di data a	*- ambil elemen a_1 di data a*	*- ambil elemen a_2 di data a*	*- ambil elemen a_3 di data a*	*- ambil elemen a_4 di data a*	*- ambil elemen a_5 di data a*	*- ambil elemen a_6 di data a*	*- ambil elemen a_7 di data a*
- ambil elemen b_0 di data b	*- ambil elemen b_1 di data b*	*- ambil elemen b_2 di data b*	*- ambil elemen b_3 di data b*	*- ambil elemen b_4 di data b*	*- ambil elemen b_5 di data b*	*- ambil elemen b_6 di data b*	*- ambil elemen b_7 di data b*
- tambahkan a_0 dan b_0	*- tambahkan a_1 dan b_1*	*- tambahkan a_2 dan b_2*	*- tambahkan a_3 dan b_3*	*- tambahkan a_4 dan b_4*	*- tambahkan a_5 dan b_5*	*- tambahkan a_6 dan b_6*	*- tambahkan a_7 dan b_7*
- simpan hasilnya sebagai elemen c_0 di data c	*- simpan hasilnya sebagai elemen c_1 di data c*	*- simpan hasilnya sebagai elemen c_2 di data c*	*- simpan hasilnya sebagai elemen c_3 di data c*	*- simpan hasilnya sebagai elemen c_4 di data c*	*- simpan hasilnya sebagai elemen c_5 di data c*	*- simpan hasilnya sebagai elemen c_6 di data c*	*- simpan hasilnya sebagai elemen c_7 di data c*
- exit	*- exit*	*- exit*	*- exit*	*- exit*	*- exit*	*- exit*	*- exit*

end

- 8 thread dieksekusi dalam waktu yang sama
- Tools: OpenMP, POSIX Thread, Win32 Thread

Komputasi Parallel - Multiprocessing

Aplikasi: vector_addition

process_0 *start*	***process_1*** *start*	***process_2*** *start*	***process_3*** *start*
thread_0: *- ambil elemen a_0 di data a* *- ambil elemen b_0 di data b* *- tambahkan a_0 dan b_0* *- simpan hasilnya sebagai elemen c_0 di data c* *- ambil elemen a_1 di data a* *- ambil elemen b_1 di data b* *- tambahkan a_1 dan b_1* *- simpan hasilnya sebagai elemen c_1 di data c* *- exit*	*thread_0:* *- ambil elemen a_0 di data a* *- ambil elemen b_0 di data b* *- tambahkan a_0 dan b_0* *- simpan hasilnya sebagai elemen c_0 di data c* *- ambil elemen a_1 di data a* *- ambil elemen b_1 di data b* *- tambahkan a_1 dan b_1* *- simpan hasilnya sebagai elemen c_1 di data c* *- exit*	*thread_0:* *- ambil elemen a_0 di data a* *- ambil elemen b_0 di data b* *- tambahkan a_0 dan b_0* *- simpan hasilnya sebagai elemen c_0 di data c* *- ambil elemen a_1 di data a* *- ambil elemen b_1 di data b* *- tambahkan a_1 dan b_1* *- simpan hasilnya sebagai elemen c_1 di data c* *- exit*	*thread_0:* *- ambil elemen a_0 di data a* *- ambil elemen b_0 di data b* *- tambahkan a_0 dan b_0* *- simpan hasilnya sebagai elemen c_0 di data c* *- ambil elemen a_1 di data a* *- ambil elemen b_1 di data b* *- tambahkan a_1 dan b_1* *- simpan hasilnya sebagai elemen c_1 di data c* *- exit*
end	*end*	*end*	*end*
address space	address space	address space	address space

- Dikenal dengan *Message Passing Interface (MPI)*
- Tools: OpenMPI, MPICH

- Konsep Komputasi Paralel
- Konsep Komputasi Berbasis GPU
- Metode Komputasi Berbasis GPU
- Contoh Kasus

Komputasi Berbasis GPU

- GPU sebagai CPU co-processor
- Traditional GPU Computing: Shader Programming
  - Cg, developed by NVIDIA
  - GLSL, developed by Khronos Group.
  - BrookGPU, developed by Standford University
- Modern GPU Computing Platform
  - CUDA, proposed and developed by NVIDIA
  - OpenCL, proposed by Apple, developed and maintained by Khronos Group.
  - DirectCompute, developed by Microsoft, bundled in DirectX SDK.
- *Catatan: materi selanjutnya merujuk pada arsitektur di NVIDIA GPU dan CUDA platform.*

*image source: geforce.com, freeiconspng.com, pixabay.com

CPU dan GPU

CPU

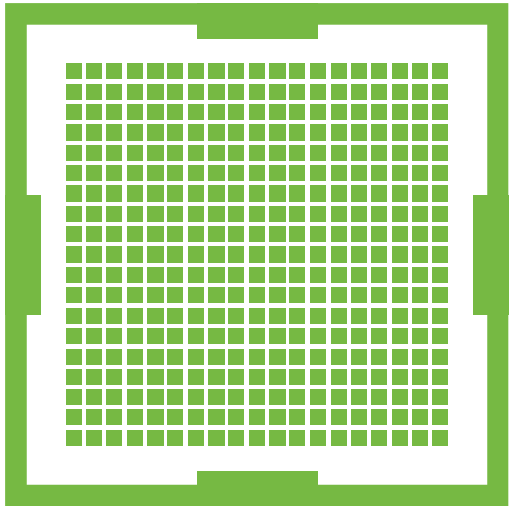
Optimized for
Serial Tasks

Low Latency

+

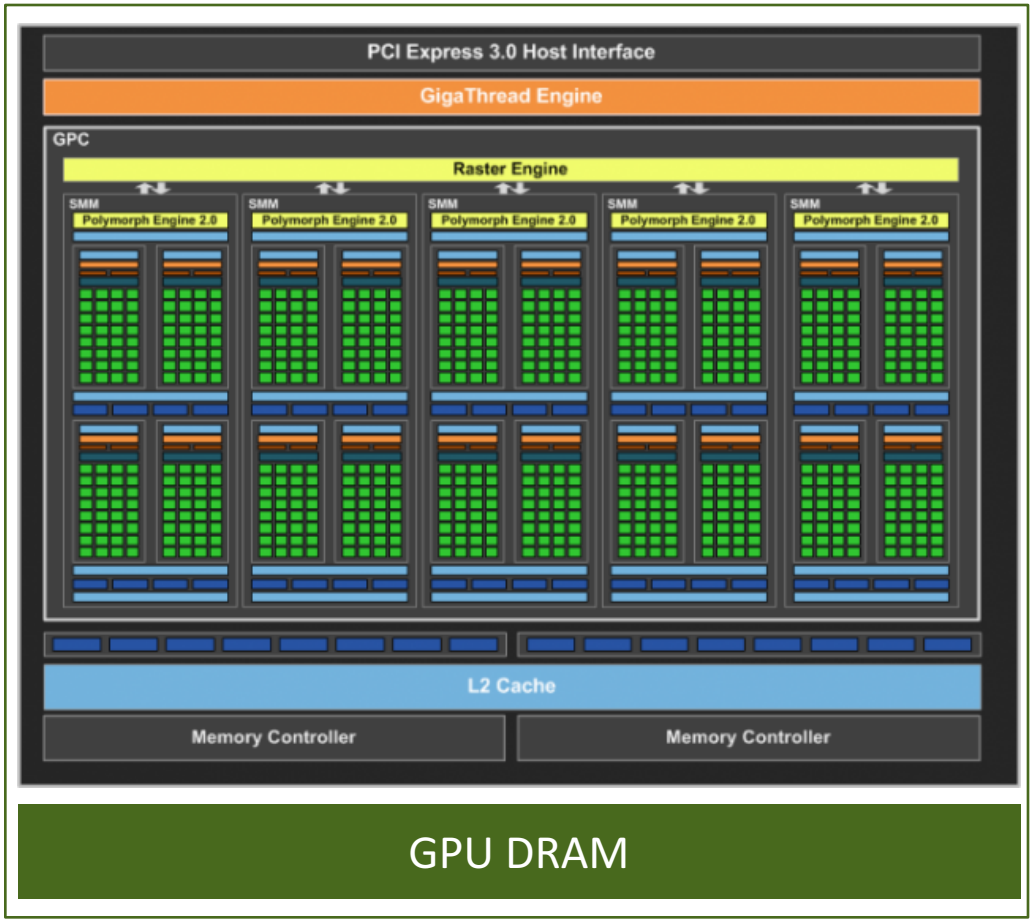
GPU Accelerator

Optimized for Many
Parallel Tasks



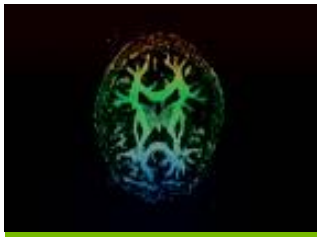
High Throughput

NVIDIA GPU Architecture – Maxwell



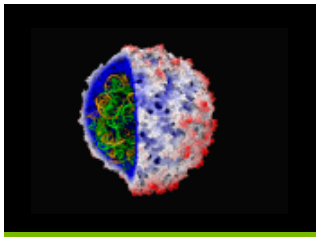
- Streaming Multiprocessor (SMM)
- L2 Cache for GPU Memory caching
- GPU DRAM (Global Memory)
- On-chip Memory

GPUs ACCELERATE SCIENCE



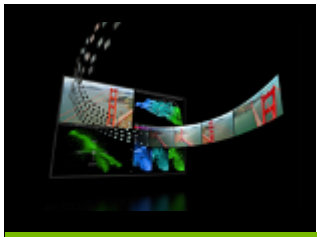
146X

Medical imaging
U of Utah



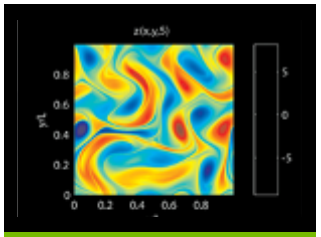
36X

Molecular Dynamics
U of Illinois, Urbana



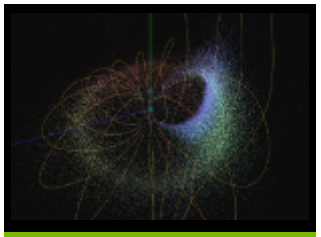
18X

Video Transcoding
Elemental Tech



50X

Matlab Computing
AccelerEyes

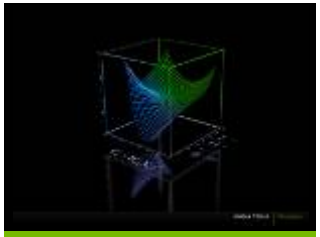


100X

Astrophysics
RIKEN

149X

Financial Simulation
Oxford



47X

Linear Algebra
Universidad Jaime

20X

3D Ultrasound
Techniscan

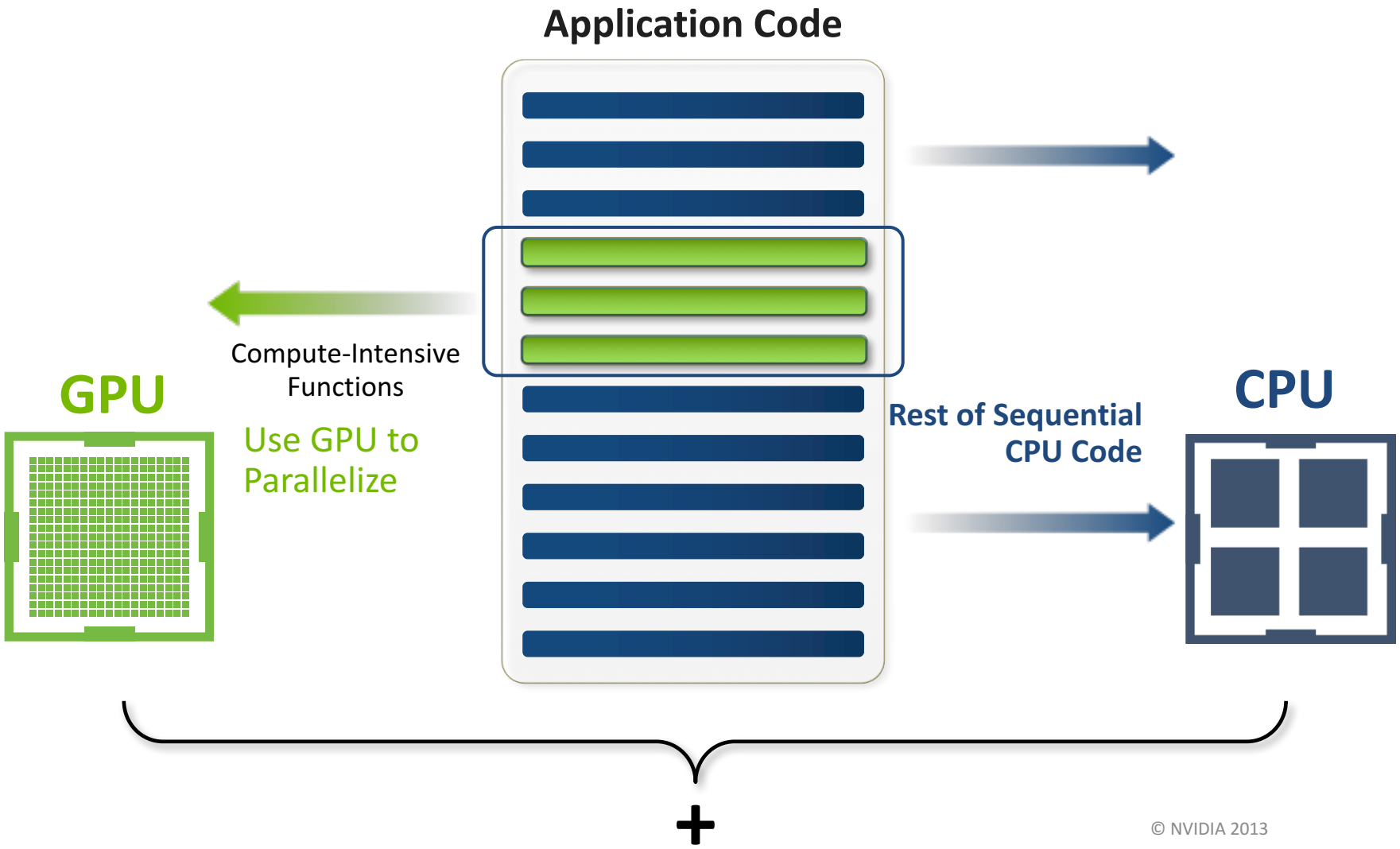
130X

Quantum Chemistry
U of Illinois, Urbana

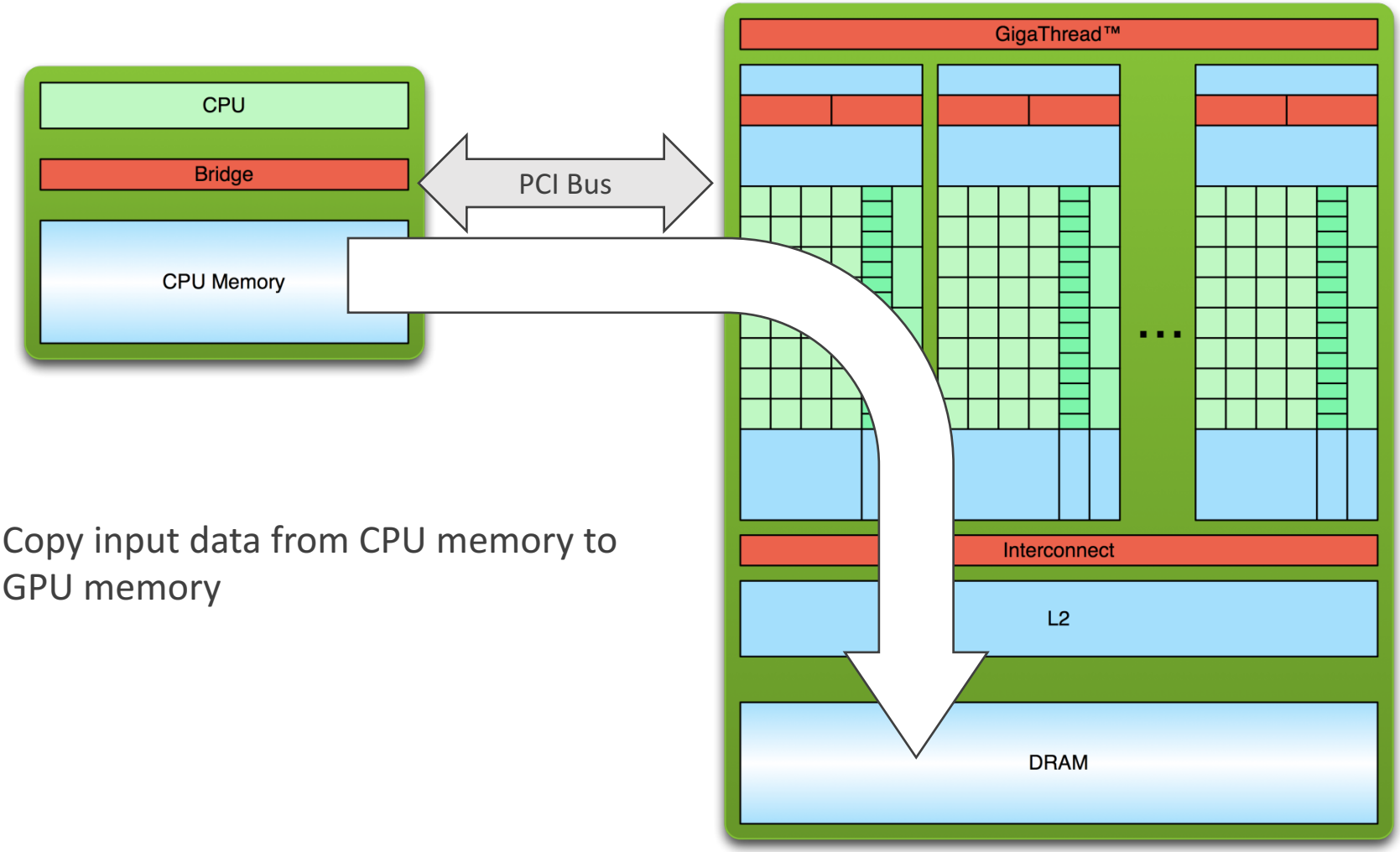
30X

Gene Sequencing
U of Maryland

Konsep Implementasi *GPU Computing* – 1



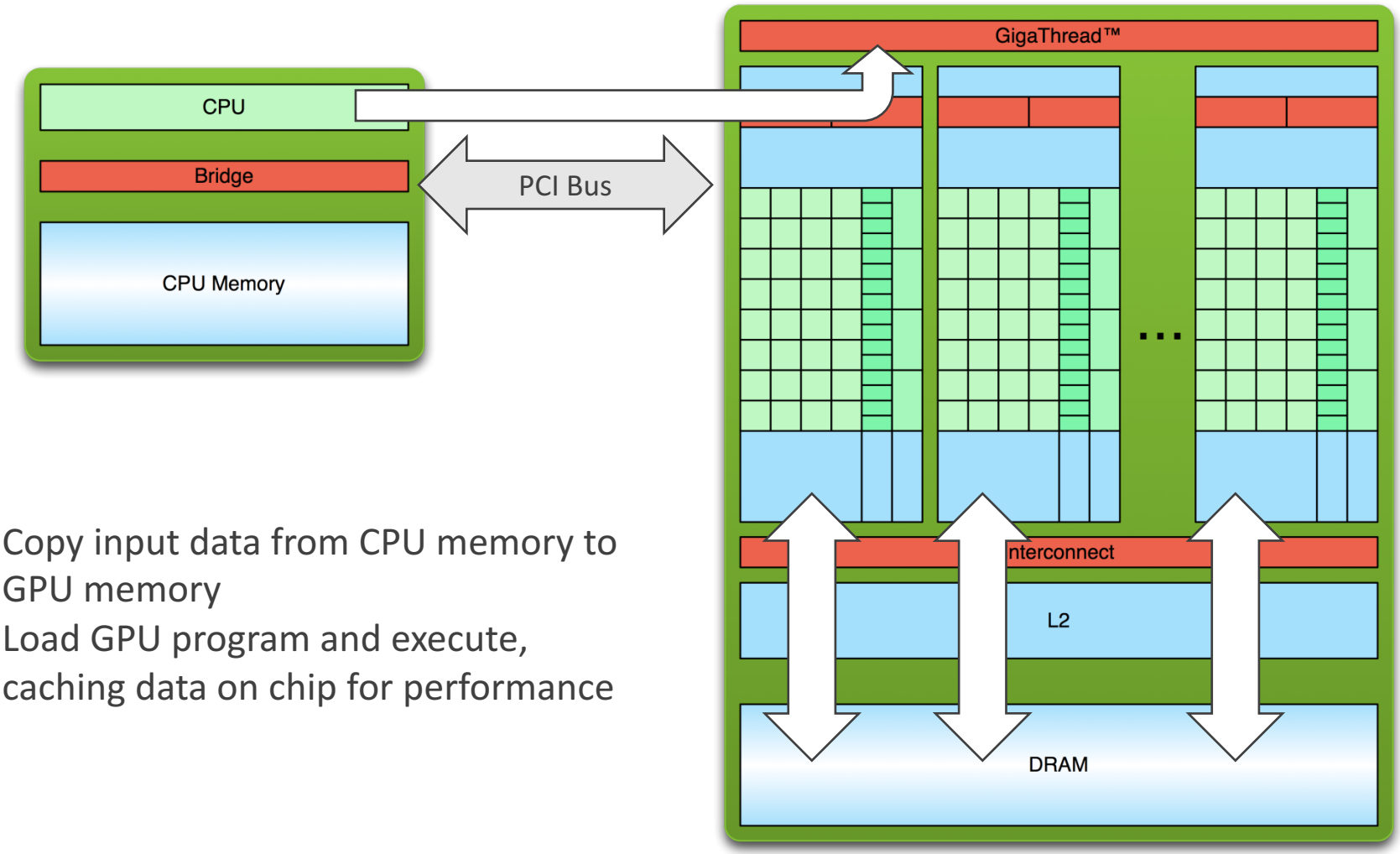
Konsep Implementasi *GPU Computing* – 2



1. Copy input data from CPU memory to GPU memory

Contents credit to NVIDIA

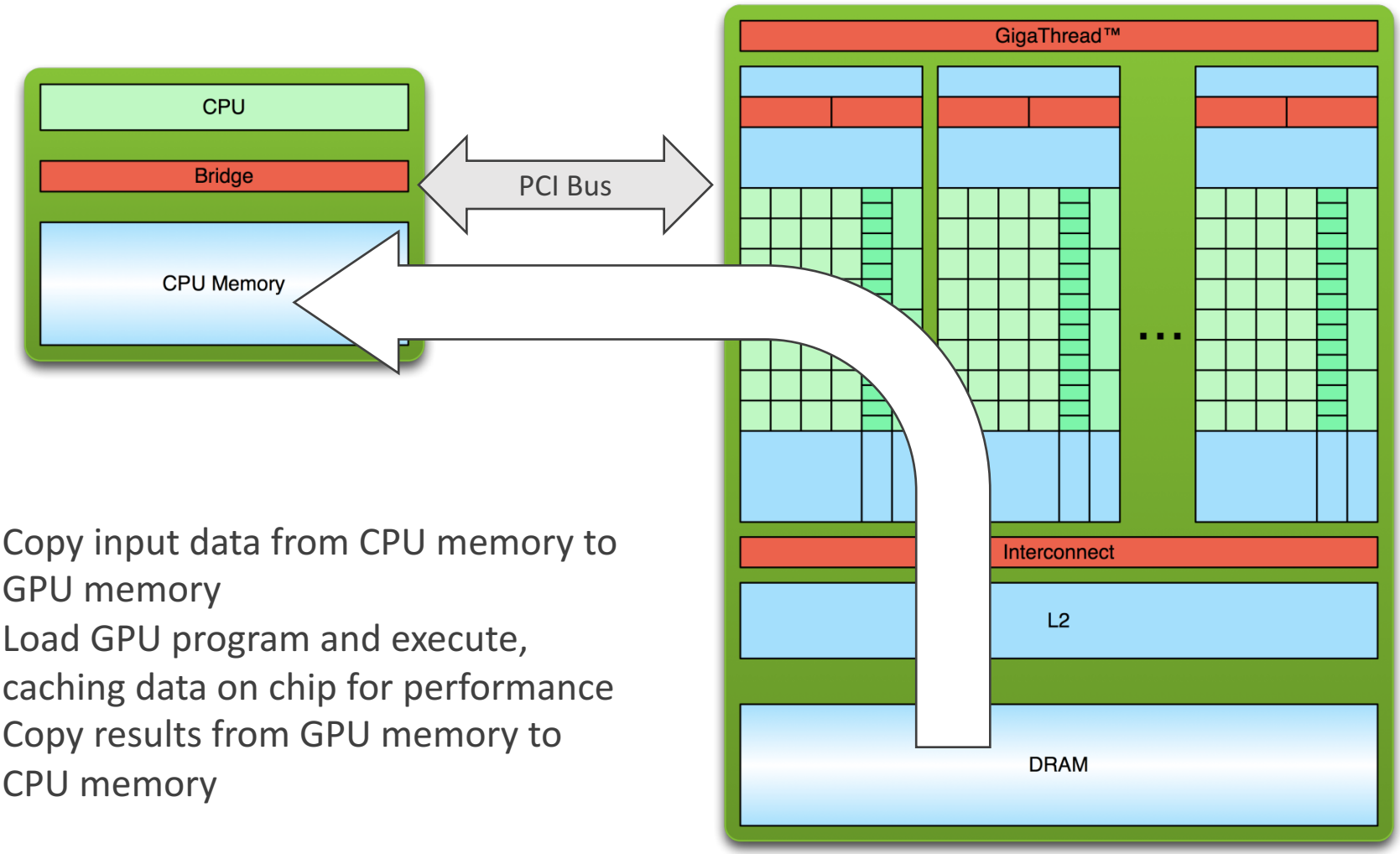
Konsep Implementasi *GPU Computing* – 3



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

Contents credit to NVIDIA

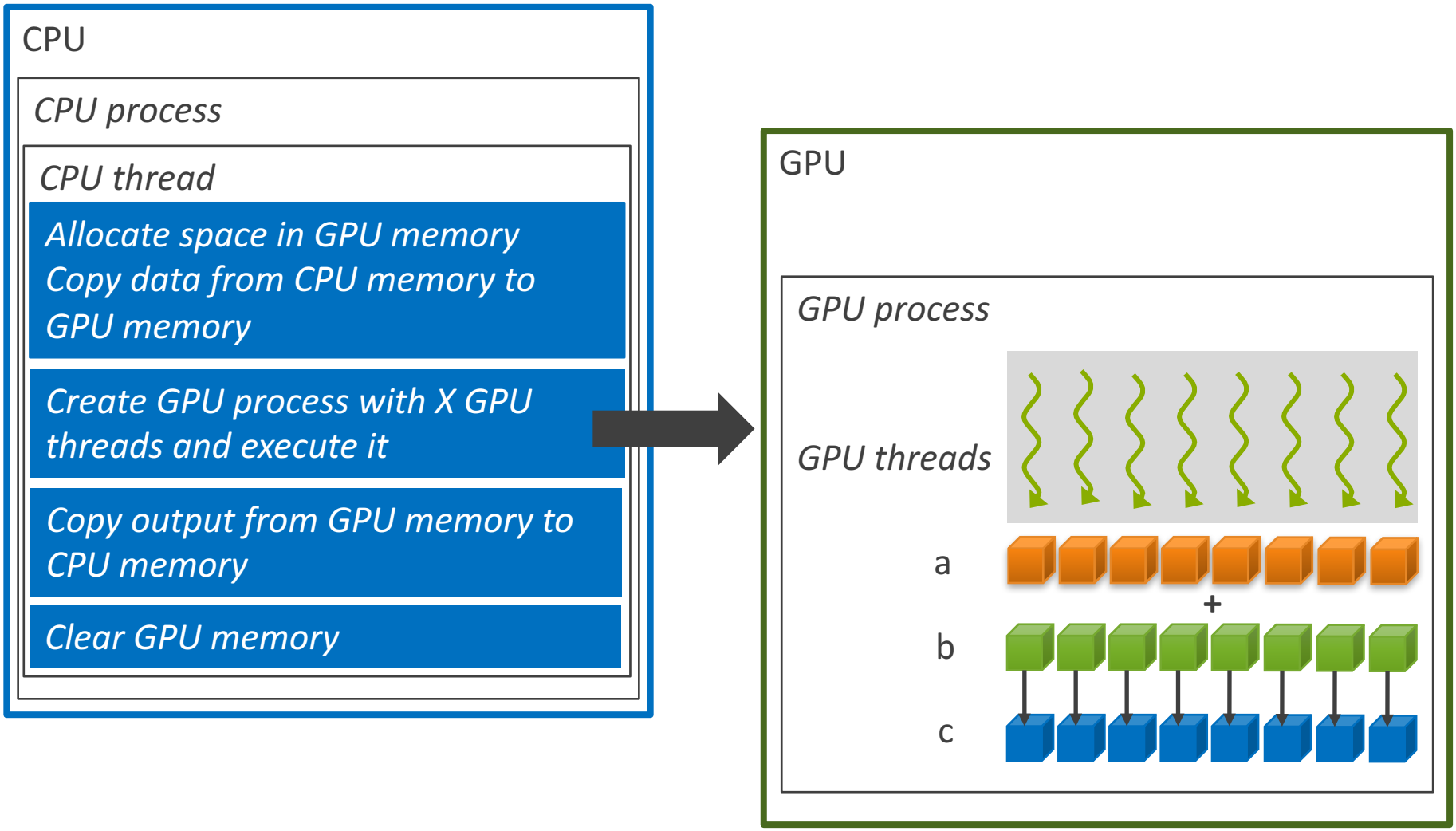
Konsep Implementasi *GPU Computing* – 4



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

Contents credit to NVIDIA

Konsep Implementasi *GPU Computing* – 5



- Konsep Komputasi Paralel
- Konsep Komputasi Berbasis GPU
- Metode Komputasi Berbasis GPU
- Contoh Kasus

Hal-hal Yang Dibutuhkan

- GPU
- GPU Driver
- CUDA Toolkit

Cara Memanfaatkan *GPU Computing*

Owned Application

Libraries	OpenACC Directives	Programming Languages	GPU-ready Applications
"Drop-in" Acceleration	Easily Accelerate Applications	Maximum Flexibility	MATLAB, Mathematica, dll

GPU ACCELERATED LIBRARIES

"Drop-in" Acceleration for Your Applications

Domain-specific

Deep Learning, GIS, EDA, Bioinformatics, Fluids

cuDNN

NVBIO

gpp Geometry Performance Primitives

Sundog Software Triton Ocean SDK

Visual Processing

Image & Video

NVIDIA CODEC SDK

FFmpeg

OpenCV

NVIDIA NPP

Linear Algebra

Dense, Sparse, Matrix

SuiteSparse CHOLMOD

MAGMA

[HiPLAR]

NVIDIA cuBLAS, cuSPARSE

Math Algorithms

AMG, Templates, Solvers

AmgX

IMSL RogueWave

Thrust & ArrayFire

NVIDIA cuRAND cuSOLVER

developer.nvidia.com/gpu-accelerated-libraries

Directive-based Programming Model

OPENACC

More Science, Less Programming

```
main()
{
  <serial code>
  #pragma acc kernels
 // automatically runs
 // on GPUs
  {
    <parallel code>
  }
}
```

SIMPLE	Minimum efforts Small code modifications
POWERFUL	Up to 10x faster application performance
PORTABLE	Optimize once, run on GPUs and CPUs

Free workstation license for academia

developer.nvidia.com/openacc

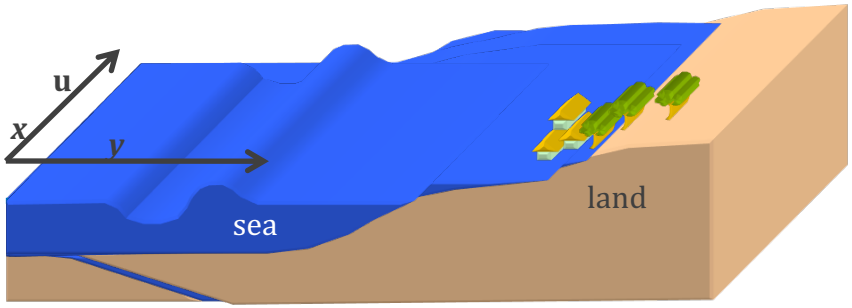
PROGRAMMING LANGUAGES

Numerical analytics	MATLAB, Mathematica, LabVIEW
Fortran	OpenACC, CUDA Fortran
C	OpenACC, CUDA C
C++	Thrust, CUDA C++
Python	Anaconda, PyCUDA
C#	Alea, Hybridizer

developer.nvidia.com/language-solutions

- Konsep Komputasi Paralel
- Konsep Komputasi Berbasis GPU
- Bagaimana Menggunakannya?
- Contoh Kasus

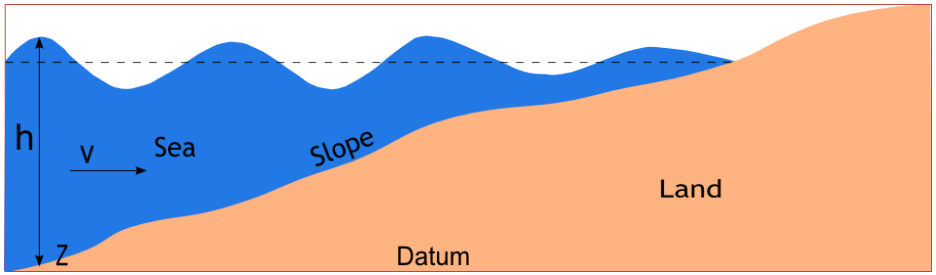
Tsunami Propagation Model – 1



$$h_t + (hu)_x + (hv)_y = 0$$

$$(uh)_t + \left(u^2h + \frac{1}{2}gh^2\right)_x + (uhv)_y = -gh(So_x - Sf_x)$$

$$(vh)_t + (uhv)_x + \left(v^2h + \frac{1}{2}gh^2\right)_y = -gh(So_y - Sf_y)$$



$$Sf_x = \frac{\eta^2 u\sqrt{u^2+v^2}}{h^{\frac{4}{3}}} \qquad Sf_y = \frac{\eta^2 v\sqrt{u^2+v^2}}{h^{\frac{4}{3}}}$$

η is equal to 0.05, as a representation of light brush surface over floodplain

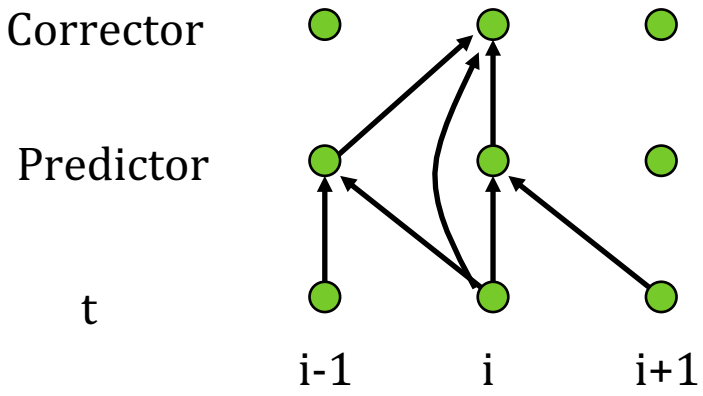
$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + g\frac{\partial h}{\partial x} = -g(So_x - Sf_x)$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + g\frac{\partial h}{\partial y} = -g(So_y - Sf_y)$$

Tsunami Propagation Model – 2

- Scheme



- Predictor

$$h_{i,j}^{t+1} = h_{i,j}^{t} - \frac{\Delta t}{\Delta x}\left(u_{i+1,j}^{t} h_{i+1,j}^{t} - u_{i,j}^{t} h_{i,j}^{t}\right) - \frac{\Delta t}{\Delta y}\left(v_{i,j+1}^{t} h_{i,j+1}^{t} - v_{i,j}^{t} h_{i,j}^{t}\right)$$

$$u_{i,j}^{t+1} = u_{i,j}^{t} - u_{i,j}^{t}\frac{\Delta t}{\Delta x}\left(u_{i+1,j}^{t} - u_{i,j}^{t}\right) - v_{i,j}^{t}\frac{\Delta t}{\Delta y}\left(u_{i,j+1}^{t} - u_{i,j}^{t}\right)$$
$$-g\frac{\Delta t}{\Delta x}\left(h_{i+1,j}^{t} - h_{i,j}^{t}\right) - g\Delta t\left(So_x - Sf_x\right)$$

$$v_{i,j}^{t+1} = v_{i,j}^{t} - u_{i,j}^{t}\frac{\Delta t}{\Delta x}\left(v_{i+1,j}^{t} - v_{i,j}^{t}\right) - v_{i,j}^{t}\frac{\Delta t}{\Delta y}\left(v_{i,j+1}^{t} - v_{i,j}^{t}\right)$$
$$-g\frac{\Delta t}{\Delta y}\left(h_{i,j+1}^{t} - h_{i,j}^{t}\right) - g\Delta t\left(So_y - Sf_y\right)$$

Tsunami Propagation Model – 3

- Corrector

$$h_{i,j}^{t+1} = \frac{\left(h_{i,j}^{t} + h_{i,j}^{t+1}\right)}{2} - \frac{\Delta t}{2\Delta x}\left(u_{i,j}^{t+1} h_{i,j}^{t+1} - u_{i-1,j}^{t+1} h_{i-1,j}^{t+1}\right) - \frac{\Delta t}{2\Delta y}\left(v_{i,j}^{t+1} h_{i,j}^{t+1} - v_{i,j-1}^{t+1} h_{i,j-1}^{t+1}\right)$$

$$u_{i,j}^{t+1} = \frac{\left(u_{i,j}^{t} + u_{i,j}^{t+1}\right)}{2} - u_{i,j}^{t+1}\frac{\Delta t}{2\Delta x}\left(u_{i,j}^{t+1} - u_{i-1,j}^{t+1}\right) - v_{i,j}^{t+1}\frac{\Delta t}{2\Delta y}\left(u_{i,j}^{t+1} - u_{i,j-1}^{t+1}\right) - g\frac{\Delta t}{2\Delta x}\left(h_{i,j}^{t+1} - h_{i-1,j}^{t+1}\right) - g\frac{\Delta t}{2}\left(So_x - Sf_x\right)$$

$$v_{i,j}^{t+1} = \frac{\left(v_{i,j}^{t} + v_{i,j}^{t+1}\right)}{2} - u_{i,j}^{t+1}\frac{\Delta t}{2\Delta x}\left(v_{i,j}^{t+1} - v_{i-1,j}^{t+1}\right) - v_{i,j}^{t+1}\frac{\Delta t}{2\Delta y}\left(v_{i,j}^{t+1} - v_{i,j-1}^{t+1}\right) - g\frac{\Delta t}{2\Delta y}\left(h_{i,j}^{t+1} - h_{i,j-1}^{t+1}\right) - g\frac{\Delta t}{2}\left(So_y - Sf_y\right)$$

Tsunami Propagation Model – 4

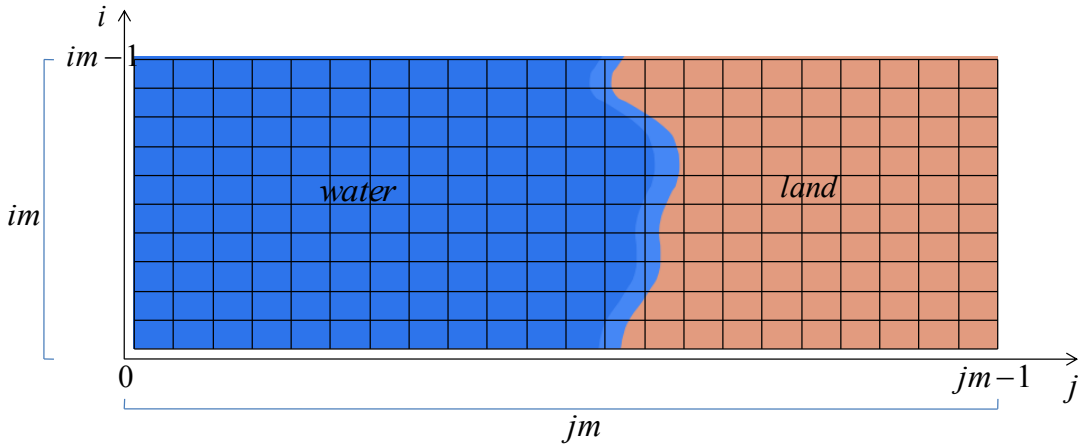
- Boundary Condition
  - Neumann boundary → for *h* and velocity @corresponding axis.
  - Copy boundary → for the velocity perpendicular to the axis.
  - Wet-dry condition→ for the wave front → by giving a tolerance value. If *h* is lower than the tolerance, then it set to be dry.
- Hansen filtering method

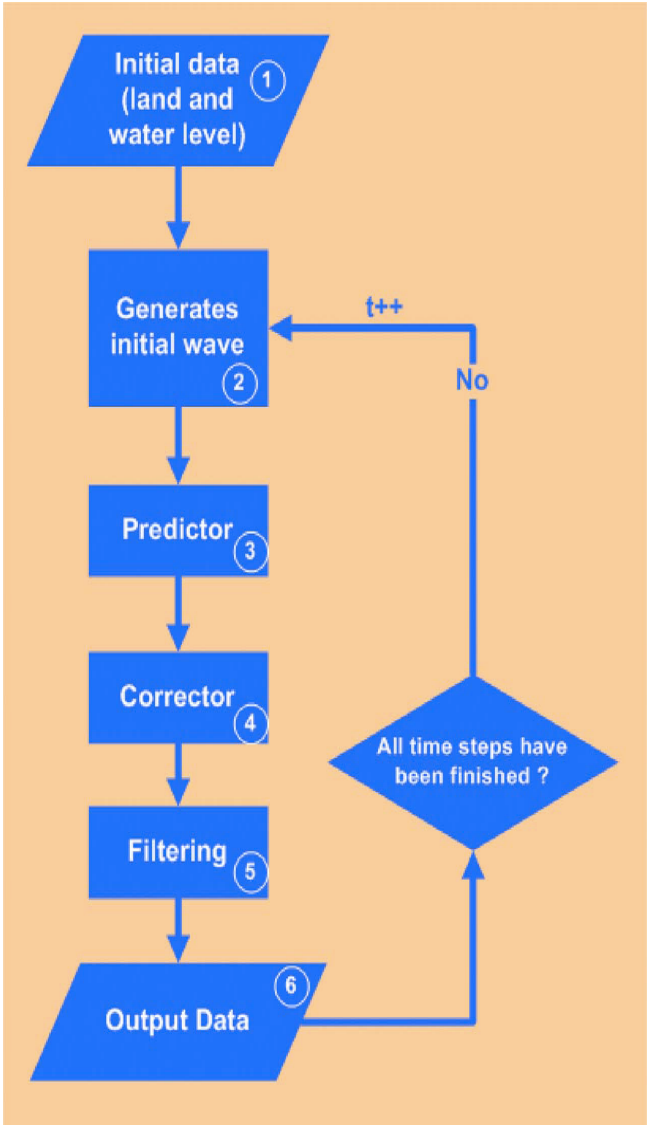
$$F_{i,j} = cF_{i,j} + (1-c)(0.25)\left(F_{i-1,j} + F_{i+1,j} + F_{i,j-1} + F_{i,j+1}\right)$$

where *F* represents the values of *h*, *u*, and *v*, and *c* is equal to 0.99.

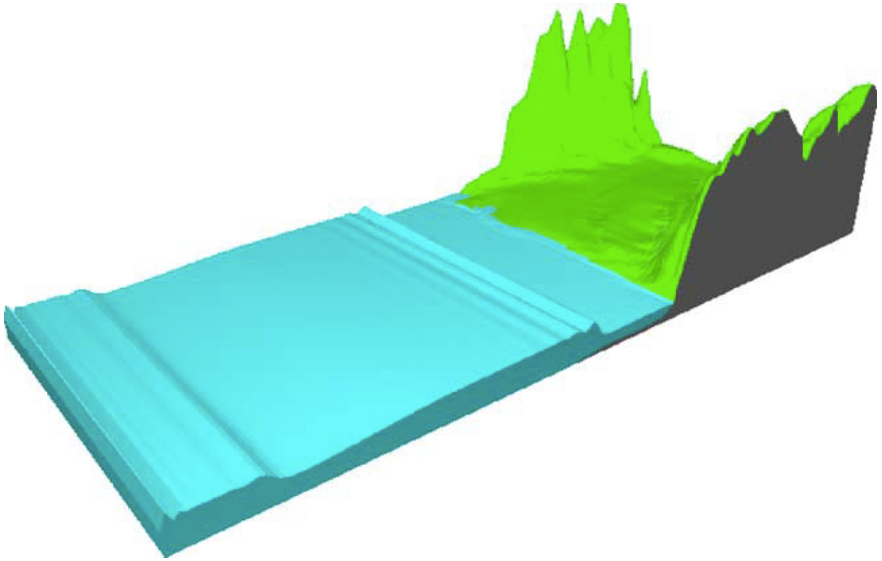
- Wave Generation
  - *h* is generated by an interpolation of water level data → *spline interpolation*.
  - $u = 0$.
  - $v = \sqrt{gh}$

Simulation Flow





Performance



Performance on NVIDIA Fermi C2050 GPU with 448 cores

Number of Time Steps	Execution Time (ms)	Speedup *
100	0.80	422x
500	3.68	473x
1000	11.47	223x

* Compared to single core on Intel Xeon E5620 2.4 GHz

Office

NovaGlobal Pte Ltd
Green & Scientific Computing Solutions
http://novaglobal.com.sg

- Established in 1996
- Operating in ASEAN region
  - Primarily in Singapore, Malaysia, Indonesia, Thailand and Vietnam
- Solution provider for
  - High Performance Computing
  - Accelerated GPU Computing
  - Cloud/Virtualization
- Platform-Independent
- Partnering with Technology Leaders

NVIDIA PREFERRED SOLUTION PROVIDER

IBM Business Partner

ScaleMP™

Adaptive COMPUTING

Intel® Solutions for Lustre* Reseller

* Other names and brands may be claimed as the property of others

intel