Introduction to GPU architecture

Caroline Collange
Inria Rennes – Bretagne Atlantique

caroline.collange@inria.fr
https://team.inria.fr/pacap/members/collange/

Master 2 SIF
ADA - 2019
Graphics processing unit (GPU)

- Graphics rendering accelerator for computer games
  - Mass market: low unit price, amortized R&D
  - Increasing programmability and flexibility
- Inexpensive, high-performance parallel processor
  - GPUs are everywhere, from cell phones to supercomputers
- General-Purpose computation on GPU (GPGPU)
GPUs in high-performance computing

- GPU/accelerator share in Top500 supercomputers
  - In 2010: 2%
  - In 2018: 22%

- 2016+ trend:
  Heterogeneous multi-core processors influenced by GPUs

#1 Summit (USA)
4,608 × (2 Power9 CPUs + 6 Volta GPUs)

#3 Sunway TaihuLight (China)
40,960 × SW26010 (4 big + 256 small cores)
GPUs in the future?

- **Yesterday (2000-2010)**
  - Homogeneous multi-core
  - Discrete components

- **Today (2011-...)**
  **Chip-level integration**
  - CPU cores and GPU cores on the same chip
  - Still different programming models, software stacks

- **Tomorrow**
  **Heterogeneous multi-core**
  - GPUs to blend into throughput-optimized, general purpose cores?
Outline

- GPU, many-core: why, what for?
  - Technological trends and constraints
  - From graphics to general purpose
  - Hardware trends

- Forms of parallelism, how to exploit them
  - Why we need (so much) parallelism: latency and throughput
  - Sources of parallelism: ILP, TLP, DLP
  - Uses of parallelism: horizontal, vertical

- Let's design a GPU!
  - Ingredients: Sequential core, Multi-core, Multi-threaded core, SIMD
  - Putting it all together
  - Architecture of current GPUs: cores, memory
The free lunch era... was yesterday

- 1980's to 2002: *Moore's law, Dennard scaling*, micro-architecture improvements
  - Exponential performance increase
  - Software compatibility preserved

Hennessy, Patterson. Computer Architecture, a quantitative approach. 5th Ed. 2010

- Do not rewrite software, buy a new machine!
Technology evolution

- **Memory wall**
  - Memory speed does not increase as fast as computing speed
  - Harder to hide memory latency

- **Power wall**
  - Power consumption of transistors does not decrease as fast as density increases
  - Performance is now limited by power consumption

- **ILP wall**
  - Law of diminishing returns on Instruction-Level Parallelism
  - Pollack rule: cost $\approx$ performance$^2$
Usage changes

- New applications demand **parallel processing**
  - Computer games: 3D graphics
  - Search engines, social networks… “big data” processing

- New computing devices are **power-constrained**
  - Laptops, cell phones, tablets…
    - Small, light, battery-powered
  - Datacenters
    - High power supply and cooling costs
Latency vs. throughput

- **Latency**: time to solution
  - Minimize time, at the expense of power
  - Metric: time
e.g. seconds

- **Throughput**: quantity of tasks processed per unit of time
  - Assumess unlimited parallelism
  - Minimize energy per operation
  - Metric: operations / time
e.g. Gflops / s

- CPU: optimized for latency
- GPU: optimized for throughput
Amdahl's law

- Bounds speedup attainable on a parallel machine

\[ S = \frac{1}{(1 - P) + \frac{P}{N}} \]

- Time to run sequential portions
- Time to run parallel portions

\( S \) Speedup
\( P \) Ratio of parallel portions
\( N \) Number of processors

Why heterogeneous architectures?

Latency-optimized multi-core (CPU)
- Low efficiency on parallel portions: spends too much resources

Throughput-optimized multi-core (GPU)
- Low performance on sequential portions

Heterogeneous multi-core (CPU+GPU)
- Use the right tool for the right job
- Allows aggressive optimization for latency or for throughput

\[ S = \frac{1}{(1-P)^t + \frac{P}{N}} \]

Example: System on Chip for smartphone

- **Big cores** for applications
- **Tiny cores** for background activity
- **GPU**
- **Special-purpose accelerators**
- **Lots of interfaces**

Diagram showing various components and interfaces of a System on Chip (SoC) for a smartphone, including LPDDR2, LPDDR2, NAND/NOR Flash, SD, eMMC, SSD, USB SS/HS host/target, 3x USB 2.0 host (ULPI/TLL/HSIC), MIPI LLI/UniPort™-M, MIPI CSI-3, 3x MIPI CSI-2 + CPI, Companion device, Camera control, DIG MIC, USB HS target, M-Shield system security technology, ARM Cortex-M4, ARM Cortex-A15 MPCore, and various interfaces such as HDMI 1.4a, LCD, MIPI DSI.
Outline

- GPU, many-core: why, what for?
  - Technological trends and constraints
  - From graphics to general purpose
  - Hardware trends

- Forms of parallelism, how to exploit them
  - Why we need (so much) parallelism: latency and throughput
  - Sources of parallelism: ILP, TLP, DLP
  - Uses of parallelism: horizontal, vertical

- Let's design a GPU!
  - Ingredients: Sequential core, Multi-core, Multi-threaded core, SIMD
  - Putting it all together
  - Architecture of current GPUs: cores, memory
The (simplest) graphics rendering pipeline

- **Vertices**
  - (simplest) graphics rendering pipeline
    - Clipping, Rasterization
    - Attribute interpolation
  - Primitives (triangles…)
- **Vertices**
- **Fragments**
  - Vertex shader
  - Clipping, Rasterization
  - Attribute interpolation
- **Fragments**
- **Framebuffer**
  - Fragment shader
  - Z-Compare
  - Blending
- **Pixels**
  - Textures
  - Programmable stage
  - Parametrizable stage
How much performance do we need

... to run 3DMark 11 at 50 frames/second?

<table>
<thead>
<tr>
<th>Element</th>
<th>Per frame</th>
<th>Per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>12.0M</td>
<td>600M</td>
</tr>
<tr>
<td>Primitives</td>
<td>12.6M</td>
<td>630M</td>
</tr>
<tr>
<td>Fragments</td>
<td>180M</td>
<td>9.0G</td>
</tr>
<tr>
<td>Instructions</td>
<td>14.4G</td>
<td>720G</td>
</tr>
</tbody>
</table>

- Intel Core i7 2700K: 56 Ginsn/s peak
  - We need to go 13x faster
  - Make a special-purpose accelerator

Source: Damien Triolet, Hardware.fr
Aside: GPU as an out-of-order pipeline

- Graphics APIs demand that primitives are drawn in submission order
  - e.g. back-to-front rendering
- Shaders proceed out of order
  - 10 000s fragments in flight
  - Shaders render fragments out of order
  - Raster ops put fragments back in order for framebuffer update
  - Various binning and tiling techniques to identify independent fragments

- General-purpose compute pipeline is much *simpler* than graphics pipeline
GPGPU: General-Purpose computation on GPUs

GPGPU history summary

Microsoft DirectX

NVIDIA

NV10 NV20 NV30 NV40 G70 G80-G90 GT200 GF100

FP 16 Programmable shaders FP 32 Dynamic control flow SIMT CUDA

ATI/AMD

R100 R200 R300 R400 R500 R600 R700 Evergreen

CTM FP 64 CAL

GPGPU traction

2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
Today: what do we need GPUs for?

1. 3D graphics rendering for games
   ♦ Complex texture mapping, lighting computations…

2. Computer Aided Design workstations
   ♦ Complex geometry

3. High-performance computing
   ♦ Complex synchronization, off-chip data movement, high precision

4. Convolutional neural networks
   ♦ Complex scheduling of low-precision linear algebra

♦ One chip to rule them all
   ♦ Find the common denominator
Outline

- GPU, many-core: why, what for?
  - Technological trends and constraints
  - From graphics to general purpose
  - Hardware trends

- Forms of parallelism, how to exploit them
  - Why we need (so much) parallelism: latency and throughput
  - Sources of parallelism: ILP, TLP, DLP
  - Uses of parallelism: horizontal, vertical

- Let's design a GPU!
  - Ingredients: Sequential core, Multi-core, Multi-threaded core, SIMD
  - Putting it all together
  - Architecture of current GPUs: cores, memory
Trends: compute performance

Caveat: only considers desktop CPUs. Gap with server CPUs is “only” 4×!
Trends: memory bandwidth

![Graph showing trends in memory bandwidth over generations. The graph compares different generations of NVIDIA GPUs, AMD GPUs, Intel CPUs, and Intel MIC. Key features include HBM, GDDR5x, and an integrated memory controller.]
Trends: energy efficiency

Energy efficiency (GFlops/W)

Year


NVIDIA GPU
AMD GPU
Intel CPU
Intel MIC

Generation

RTX 2080 Ti
RX Vega 64
Titan X
R9 Fury X
GTX 980
K20X
R9 290X
GTX 780
GT650
GT580
GTX 480
GTX 465
GTX 460
2900 XT
9800 GX 870
8800 GTX

5×
7×
Outline

- GPU, many-core: why, what for?
  - Technological trends and constraints
  - From graphics to general purpose
  - Hardware trends

- Forms of parallelism, how to exploit them
  - Why we need (so much) parallelism: latency and throughput
  - Sources of parallelism: ILP, TLP, DLP
  - Uses of parallelism: horizontal, vertical

- Let's design a GPU!
  - Ingredients: Sequential core, Multi-core, Multi-threaded core, SIMD
  - Putting it all together
  - Architecture of current GPUs: cores, memory
What is parallelism?

Parallelism: independent operations which execution can be overlapped
Operations: memory accesses or computations

How much parallelism do I need?
- Little's law in queuing theory
  - Average customer arrival rate $\lambda \leftarrow$ throughput
  - Average time spent $W \leftarrow$ latency
  - Average number of customers $L = \lambda \times W \leftarrow$ Parallelism = throughput $\times$ latency

Units
- For memory: $B = \text{GB/s} \times \text{ns}$
- For arithmetic: $\text{flops} = \text{Gflops/s} \times \text{ns}$

J. Little. A proof for the queuing formula $L= \lambda W$. JSTOR 1961.
Throughput and latency: CPU vs. GPU

CPU memory: Core i7 4790, DDR3-1600, 2 channels
- Throughput: 25.6 GB/s
- Latency: 67 ns

GPU memory: NVIDIA GeForce GTX 980, GDDR5-7010, 256-bit
- Throughput: 224 GB/s
- Latency: 410 ns

→ Need 56 times more parallelism!
Consequence: more parallelism

- GPU vs. CPU
  - 8× more parallelism to feed more units (throughput)
  - 6× more parallelism to hide longer latency
  - 56× more total parallelism
- How to find this parallelism?
Sources of parallelism

- **ILP: Instruction-Level Parallelism**
  - Between independent instructions in sequential program

- **TLP: Thread-Level Parallelism**
  - Between independent execution contexts: threads

- **DLP: Data-Level Parallelism**
  - Between elements of a vector: same operation on several elements
Example: $X \leftarrow a \times X$

- In-place scalar-vector product: $X \leftarrow a \times X$

Sequential (ILP)

For $i = 0$ to $n-1$ do:
$X[i] \leftarrow a \times X[i]$

Threads (TLP)

Launch $n$ threads:
$X[tid] \leftarrow a \times X[tid]$

Vector (DLP)

$X \leftarrow a \times X$

- Or any combination of the above
Uses of parallelism

- “Horizontal” parallelism for throughput
  - More units working in parallel

- “Vertical” parallelism for latency hiding
  - Pipelining: keep units busy when waiting for dependencies, memory
### How to extract parallelism?

<table>
<thead>
<tr>
<th></th>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILP</strong></td>
<td>Superscalar</td>
<td>Pipelined</td>
</tr>
<tr>
<td><strong>TLP</strong></td>
<td>Multi-core SMT</td>
<td>Interleaved / switch-on-event multithreading</td>
</tr>
<tr>
<td><strong>DLP</strong></td>
<td>SIMD / SIMT</td>
<td>Vector / temporal SIMT</td>
</tr>
</tbody>
</table>

- We have seen the first row: ILP
- We will now review techniques for the next rows: TLP, DLP
Outline

- GPU, many-core: why, what for?
  - Technological trends and constraints
  - From graphics to general purpose
  - Hardware trends

- Forms of parallelism, how to exploit them
  - Why we need (so much) parallelism: latency and throughput
  - Sources of parallelism: ILP, TLP, DLP
  - Uses of parallelism: horizontal, vertical

- Let's design a GPU!
  - Ingredients: Sequential core, Multi-core, Multi-threaded core, SIMD
  - Putting it all together
  - Architecture of current GPUs: cores, memory
Sequential processor

- Focuses on instruction-level parallelism
  - Exploits ILP: vertically (pipelining) and horizontally (superscalar)
The incremental approach: multi-core

- Several processors on a single chip sharing one memory space

- Area: benefits from Moore's law
- Power: extra cores consume little when not in use
  - e.g. Intel Turbo Boost
Homogeneous multi-core

- Horizontal use of thread-level parallelism

- Improves peak throughput
Example: Tilera Tile-GX

- Grid of (up to) 72 tiles
- Each tile: 3-way VLIW processor, 5 pipeline stages, 1.2 GHz
Interleaved multi-threading

- Vertical use of thread-level parallelism

- Hides latency thanks to explicit parallelism improves achieved throughput
Example: Oracle Sparc T5

- 16 cores / chip
- Core: out-of-order superscalar, 8 threads
- 15 pipeline stages, 3.6 GHz
Clustered multi-core

- For each individual unit, select between
  - Horizontal replication
  - Vertical time-multiplexing

- Examples
  - Sun UltraSparc T2, T3
  - AMD Bulldozer
  - IBM Power 7, 8, 9

- Area-efficient tradeoff
- Blurs boundaries between cores
Implicit SIMD

- **Factorization** of fetch/decode, load-store units
  - Fetch 1 instruction on behalf of several threads
  - Read 1 memory location and broadcast to several registers

- In NVIDIA-speak
  - SIMT: Single Instruction, Multiple Threads
  - Convoy of synchronized threads: *warp*

- Extracts DLP from multi-thread applications
How to exploit common operations?

Multi-threading implementation options:

- **Horizontal**: replication
  - **Different** resources, **same** time
  - Chip Multi-Processing (CMP)

- **Vertical**: time-multiplexing
  - **Same** resource, **different** times
  - Multi-Threading (MT)

- **Factorization**
  - **If** we have common operations between threads
  - **Same** resource, **same** time
  - Single-Instruction Multi-Threading (SIMT)
Explicit SIMD

- Single Instruction Multiple Data
- Horizontal use of data level parallelism

### Examples
- Intel MIC (16-wide)
- AMD GCN GPU (16-wide×4-deep)
- Most general purpose CPUs (4-wide to 16-wide)
Outline

- GPU, many-core: why, what for?
  - Technological trends and constraints
  - From graphics to general purpose

- Forms of parallelism, how to exploit them
  - Why we need (so much) parallelism: latency and throughput
  - Sources of parallelism: ILP, TLP, DLP
  - Uses of parallelism: horizontal, vertical

- Let's design a GPU!
  - Ingredients: Sequential core, Multi-core, Multi-threaded core, SIMD
  - Putting it all together
  - Architecture of current GPUs: cores, memory
Hierarchical combination

- All of these techniques face the law of diminishing returns
  - More cores → complex interconnect, hard to maintain cache coherence
  - More threads/core → more register and cache pressure
  - Wider vectors → more performance lost to irregular control/data flow

- Both CPUs and GPUs combine various techniques
  - Superscalar execution
  - Multiple cores
  - Multiple threads/core
  - SIMD units
Example CPU: Intel Core i7

- Is a wide superscalar, but has also
  - Multicore
  - Multi-thread / core
  - SIMD units

- Up to 116 operations/cycle from 8 threads

256-bit SIMD units: AVX

Wide superscalar

Simultaneous Multi-Threading: 2 threads

4 CPU cores
Example GPU: NVIDIA GeForce GTX 980

- SIMT: warps of 32 threads
- 16 SMs / chip
- 4×32 cores / SM, 64 warps / SM

$\begin{array}{c}
\text{Warp 1} \\
\text{Warp 5} \\
\text{Core 1} \\
\text{...} \\
\text{Core 32} \\
\text{Warp 60}
\end{array}
\begin{array}{c}
\text{Warp 2} \\
\text{Warp 6} \\
\text{Core 33} \\
\text{...} \\
\text{Core 34} \\
\text{Warp 61}
\end{array}
\begin{array}{c}
\text{Warp 3} \\
\text{Warp 7} \\
\text{Core 63} \\
\text{...} \\
\text{Core 64} \\
\text{Warp 62}
\end{array}
\begin{array}{c}
\text{Warp 4} \\
\text{Warp 8} \\
\text{Core 91} \\
\text{...} \\
\text{Core 92} \\
\text{Warp 63}
\end{array}
\text{SM1}

- 4612 Gflop/s
- Up to 32768 threads in flight
### Taxonomy of parallel architectures

<table>
<thead>
<tr>
<th></th>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILP</strong></td>
<td>Superscalar / VLIW</td>
<td>Pipelined</td>
</tr>
<tr>
<td><strong>TLP</strong></td>
<td>Multi-core SMT</td>
<td>Interleaved / switch-on-event multithreading</td>
</tr>
<tr>
<td><strong>DLP</strong></td>
<td>SIMD / SIMT</td>
<td>Vector / temporal SIMT</td>
</tr>
</tbody>
</table>
Classification: multi-core

Intel Haswell

<table>
<thead>
<tr>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP</td>
<td>8</td>
</tr>
<tr>
<td>TLP</td>
<td>4</td>
</tr>
<tr>
<td>DLP</td>
<td>8</td>
</tr>
</tbody>
</table>

Fujitsu SPARC64 X

<table>
<thead>
<tr>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP</td>
<td>8</td>
</tr>
<tr>
<td>TLP</td>
<td>16</td>
</tr>
<tr>
<td>DLP</td>
<td>2</td>
</tr>
</tbody>
</table>

IBM Power 8

<table>
<thead>
<tr>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Oracle Sparc T5

<table>
<thead>
<tr>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

General-purpose multi-cores: balance ILP, TLP and DLP

Sparc T: focus on TLP
How to read the table

- Given an application with known ILP, TLP, DLP, how much throughput / latency hiding can I expect?
  - For each cell, take minimum of existing parallelism and hardware capability
  - The column-wise product gives throughput / latency hiding

<table>
<thead>
<tr>
<th></th>
<th>Sequential code</th>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP</td>
<td>10</td>
<td>min(8, 10)=8</td>
<td></td>
</tr>
<tr>
<td>TLP</td>
<td>1</td>
<td>min(4, 1)=1</td>
<td>2</td>
</tr>
<tr>
<td>DLP</td>
<td>1</td>
<td>min(8, 1)=1</td>
<td></td>
</tr>
</tbody>
</table>

Max throughput = 8×1×1 for this application
Peak throughput = 8×4×8 that can be achieved
→ Can only hope for ~3% of peak performance!
Classification: GPU and many small-core

<table>
<thead>
<tr>
<th></th>
<th>Intel MIC</th>
<th>Nvidia Kepler</th>
<th>AMD GCN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Horizontal</td>
<td>Vertical</td>
<td>Horizontal</td>
</tr>
<tr>
<td><strong>ILP</strong></td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td><strong>TLP</strong></td>
<td>60</td>
<td>16×4</td>
<td>32</td>
</tr>
<tr>
<td><strong>DLP</strong></td>
<td>16</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>SIMD</td>
<td>Cores</td>
<td>Cores×units</td>
<td>SIMD</td>
</tr>
</tbody>
</table>

**Kalray MPPA-256**

<table>
<thead>
<tr>
<th></th>
<th>Tilera Tile-GX</th>
<th>Kalray MPPA-256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>72</td>
<td>17×16</td>
</tr>
</tbody>
</table>

**Notes:**
- **GPU:** focus on DLP, TLP horizontal and vertical
- **Many small-core:** focus on horizontal TLP

**Horizontal**

**Vertical**
Outline

- GPU, many-core: why, what for?
  - Technological trends and constraints
  - From graphics to general purpose
- Forms of parallelism, how to exploit them
  - Why we need (so much) parallelism: latency and throughput
  - Sources of parallelism: ILP, TLP, DLP
  - Uses of parallelism: horizontal, vertical
- Let's design a GPU!
  - Ingredients: Sequential core, Multi-core, Multi-threaded core, SIMD
  - Putting it all together
  - Architecture of current GPUs: cores, memory
What is inside a graphics card?

NVIDIA Volta V100 GPU. Artist rendering!
External memory: discrete GPU

Classical CPU-GPU model

- Split memory spaces
- Need to transfer data explicitly
- Highest bandwidth from GPU memory
- Transfers to main memory are slower

Example configuration: Intel Core i7 4790, Nvidia GeForce GTX 980
Discrete GPU memory technology

- **GDDR5, GDDR5x**
  - Qualitatively like regular DDR
  - Optimized for high frequency at the expense of latency and cost
  - e.g. *Nvidia Titan X*: 12 chip pairs × 32-bit bus × 10 GHz → 480 GB/s

- **High-Bandwidth Memory (HBM)**
  - On-package stacked memory on silicon interposer
  - Shorter traces, wider bus, lower frequency: more energy-efficient
  - Limited capacity and high cost
  - e.g. *AMD R9 Fury X*: 4× 4-high stack × 1024-bit × 1 GHz → 512 GB/s
Maximizing memory bandwidth

Memory bandwidth is a critical resource

- Cache hierarchy reduces throughput demand on main memory
  - Bandwidth amplification
  - Less energy per access

- Hardware data compression in caches and memory
  - Lossy compression for textures (under programmer control)
  - Lossless compression for framebuffer, z-buffer…
Most GPUs today are integrated
- Same physical memory
- May support memory coherence
  - GPU can read directly from CPU caches
- More contention on external memory
GPU high-level organization

- **Processing units**
  - Streaming Multiprocessors (SM) in Nvidia jargon
  - Compute Unit (CU) in AMD's
  - Closest equivalent to a CPU core
  - Today: from 1 to 20 SMs in a GPU

- **Memory system: caches**
  - Keep frequently-accessed data
  - Reduce throughput demand on main memory
  - Managed by hardware (L1, L2) or software (Shared Memory)
Each SM is a highly-multithreaded processor

- Today: 24 to 48 warps of 32 threads each
  → ~1K threads on each SM, ~10K threads on a GPU
GPU: on-chip memory

- Conventional wisdom
  - Cache area in CPU vs. GPU according to the NVIDIA CUDA Programming Guide:

- But... if we include registers:

- GPUs have more internal memory than desktop CPUs
Registers: CPU vs. GPU

- Registers keep the contents of local variables
- Typical values

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers/thread</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Registers/core</td>
<td>256</td>
<td>65536</td>
</tr>
<tr>
<td>Read / Write ports</td>
<td>10R/5W</td>
<td>2R/1W</td>
</tr>
</tbody>
</table>

- GPU: many more registers, but made of simpler memory
The locality dilemma

- More cores → higher communication latency

- Solution 1: bigger caches
  (general-purpose multi-cores, Intel MIC)

- Solution 2: more threads / core
  (GPUs, Sparc T)
  Need extra memory for thread state
  → more registers, bigger caches

- Solution 3: programmer-managed communication
  (many small cores)

→ Bigger cores
→ More specialized
Where are we heading?

Alternatives for future many-cores

- **A unified world**
  General-purpose multi-cores continue to exploit more ILP, TLP and DLP
  Eventually replace all special-purpose many-cores

- **A specialized world**
  Varieties of many-cores continue evolving independently
  Co-existance of GPU for graphics, many-core for HPC, many-thread for servers…

- **A heterogeneous world**
  Special-purpose many-cores co-exist within the same chip
  Multi-core CPU + GPU + many-core accelerator…
Next time: SIMT control flow management

Software:

\[
\text{__global__ void scale(float a, float * X)}
\{
\text{unsigned int tid;}
\text{tid = blockIdx.x * blockDim.x + threadIdx.x;}
\text{X[tid] = a * X[tid];}
\}
\]

Architecture: multi-thread programming model

Hardware datapaths: SIMD execution units
Takeaway

- Parallelism for throughput and latency hiding
- Types of parallelism: ILP, TLP, DLP
- All modern processors exploit the 3 kinds of parallelism
- GPUs focus on Thread-level and Data-level parallelism