

# Why GPUs?

**Robert Strzodka (MPII),  
Dominik Göddecke (TUDo), Dominik Behr (AMD)**

**PPAM 2009 - Conference on Parallel Processing and  
Applied Mathematics  
Wrocław, Poland, September 13-16, 2009**

[www.gpgpu.org/ppam2009](http://www.gpgpu.org/ppam2009)

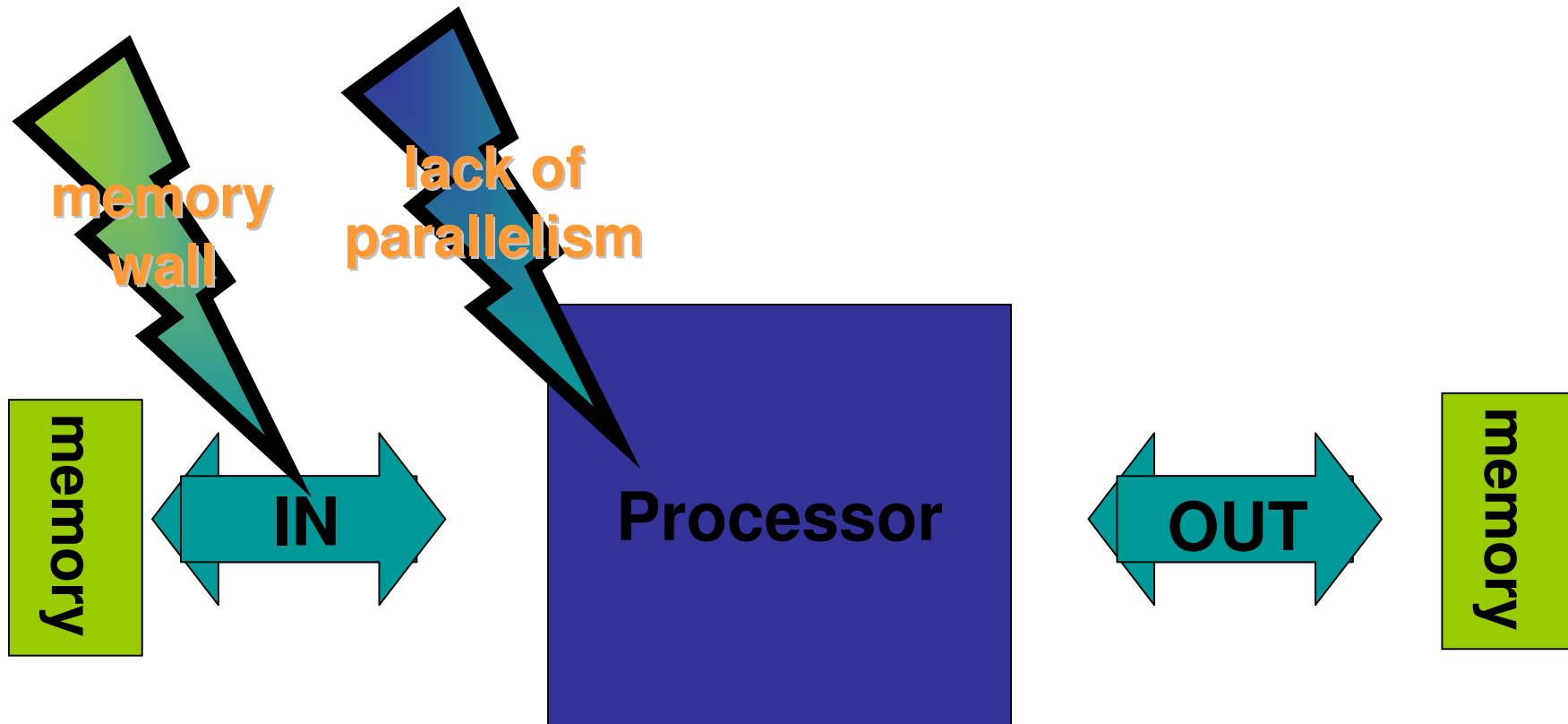
# Overview

---

- **Computation / Bandwidth / Power**
- **GPU Characteristics**

# Data Processing in General

---



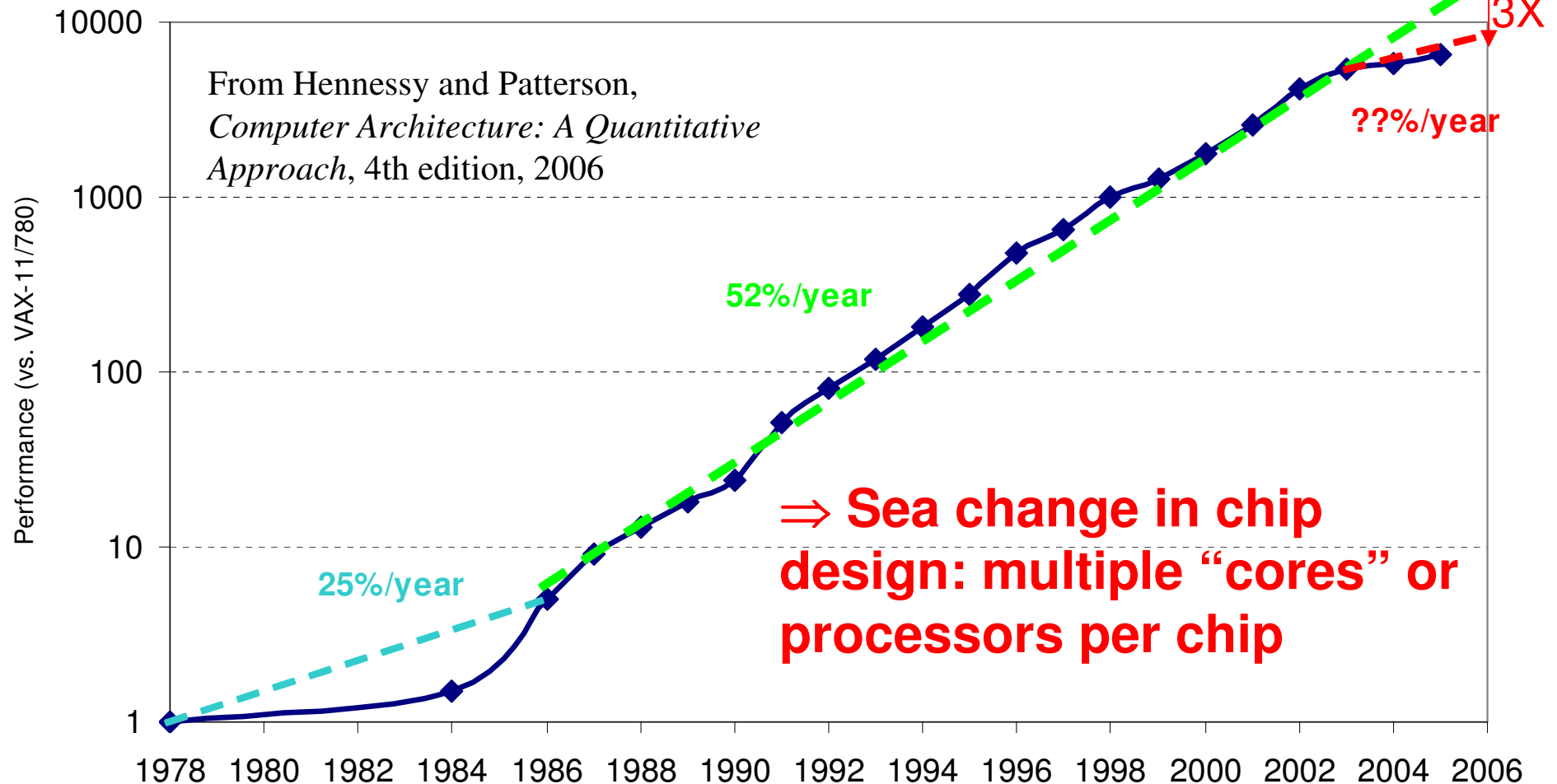
# Old and New Wisdom in Computer Architecture

---

- **Old:** Power is free, Transistors are expensive
- **New:** “Power wall”, Power expensive, Transistors free  
(Can put more transistors on chip than can afford to turn on)
  
- **Old:** Multiplies are slow, Memory access is fast
- **New:** “Memory wall”, Multiplies fast, Memory slow  
(200 clocks to DRAM memory, 4 clocks for FP multiply)
  
- **Old:** Increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, ...)
- **New:** “ILP wall”, diminishing returns on more ILP HW  
(Explicit thread and data parallelism must be exploited)
  
- **New:** Power Wall + Memory Wall + ILP Wall = Brick Wall

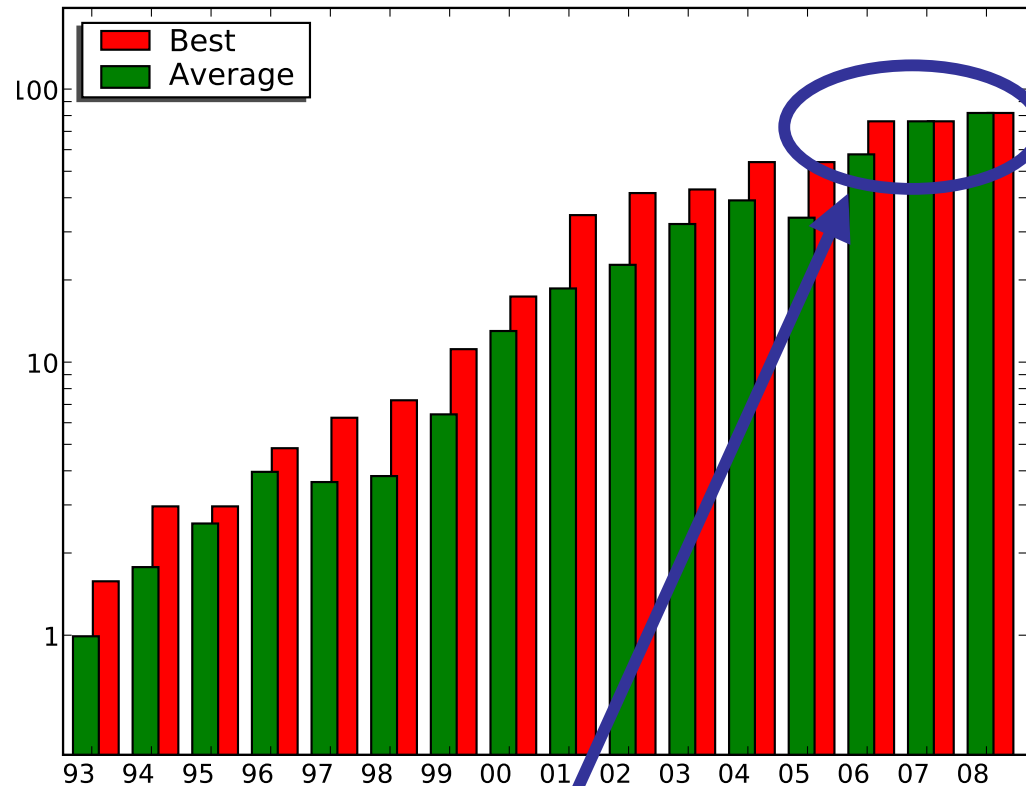
slide courtesy of  
Christos Kozyrakis

# Uniprocessor Performance (SPECint)



slide courtesy of  
Christos Kozyrakis

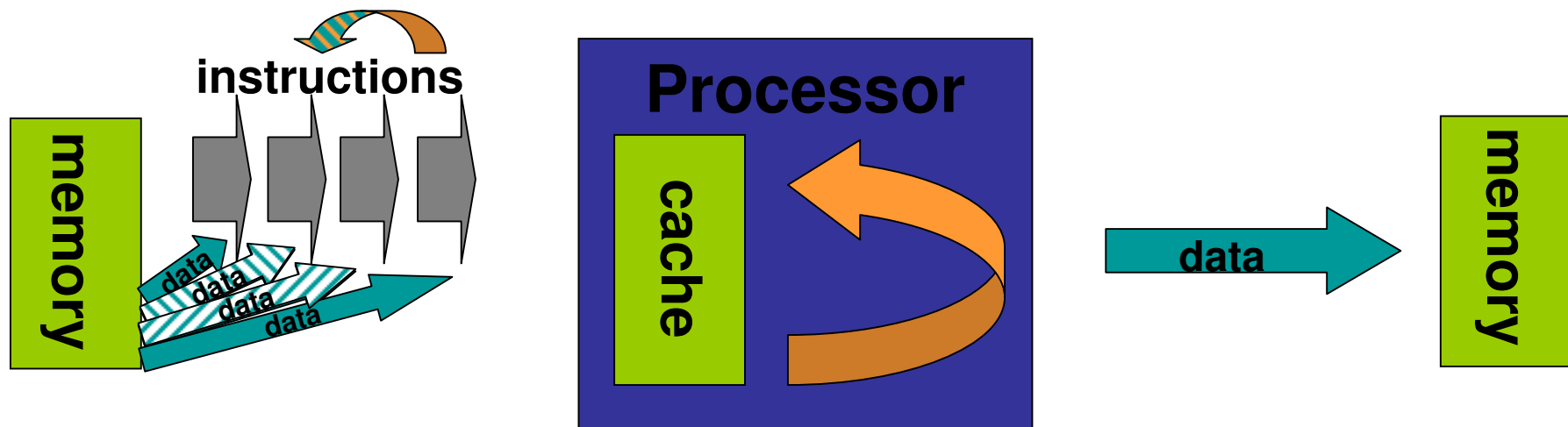
# SW Performance: FeatFlow 1993–2008



- **80x speedup in 16 years from HW for free**
- **But: HW peak performance had 1000x speedup**
- **And: Since 2006 stagnation**
- **No future for serial code: Parallelism is indispensable**

# Instruction-Stream-Based Processing

---



# Instruction- and Data-Streams

---

Addition of 2D arrays:  $C = A + B$

**instuction  
stream  
processing  
data**

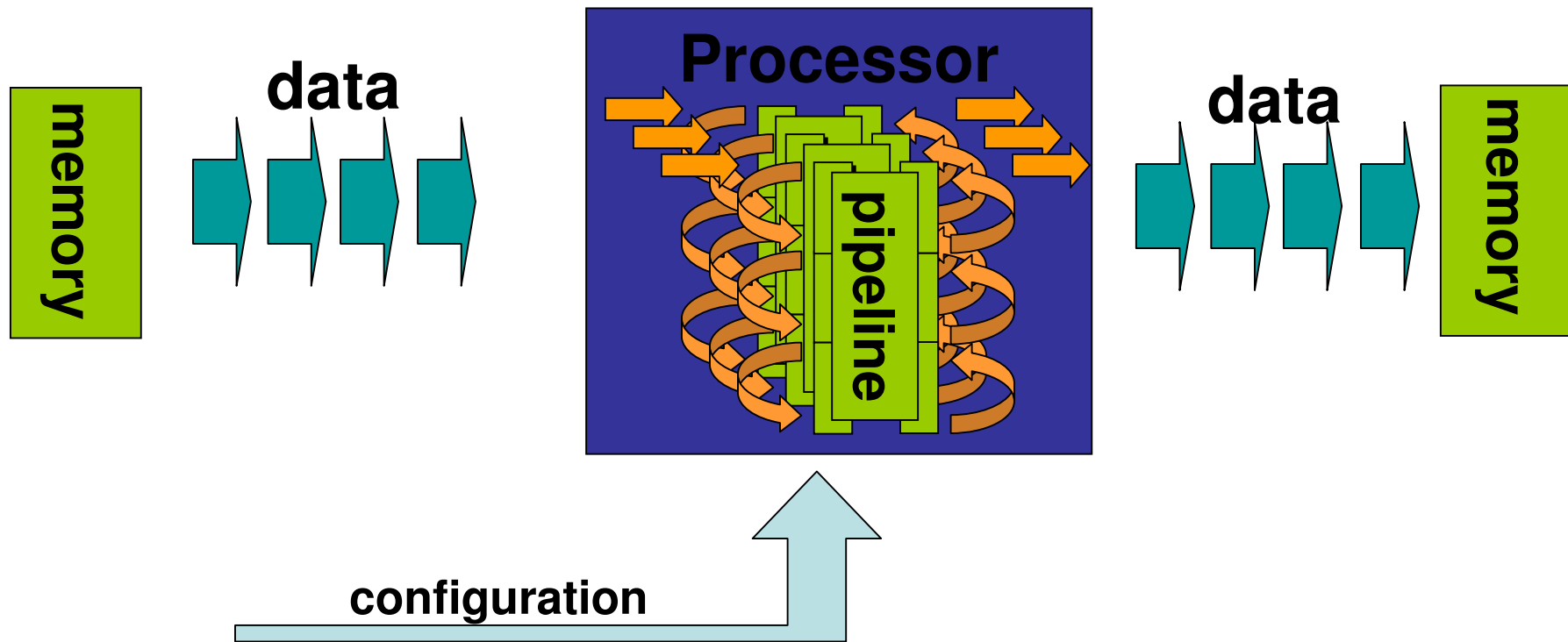
```
for (y=0; y<HEIGHT; y++)  
for (x=0; x<WIDTH; x++) {  
    C[y][x] = A[y][x] + B[y][x];  
}
```

**data streams  
undergoing a  
kernel  
operation**

```
inputStreams (A, B);  
outputStream (C);  
kernelProgram (OP_ADD);  
processStreams ();
```

# Data-Stream-Based Processing

---



# Architectures: Data – Processor Locality

---

- **Field Programmable Gate Array (FPGA)**
  - Compute by configuring Boolean functions and local memory
- **Processor Array / Multi-core Processor**
  - Assemble many (simple) processors and memories on one chip
- **Processor-in-Memory (PIM)**
  - Insert processing elements directly into RAM chips
- **Stream Processor**
  - Create data locality through a hierarchy of memories
- **Graphics Processor Unit (GPU)**
  - Hide data access latencies by keeping 1000s of threads in-flight

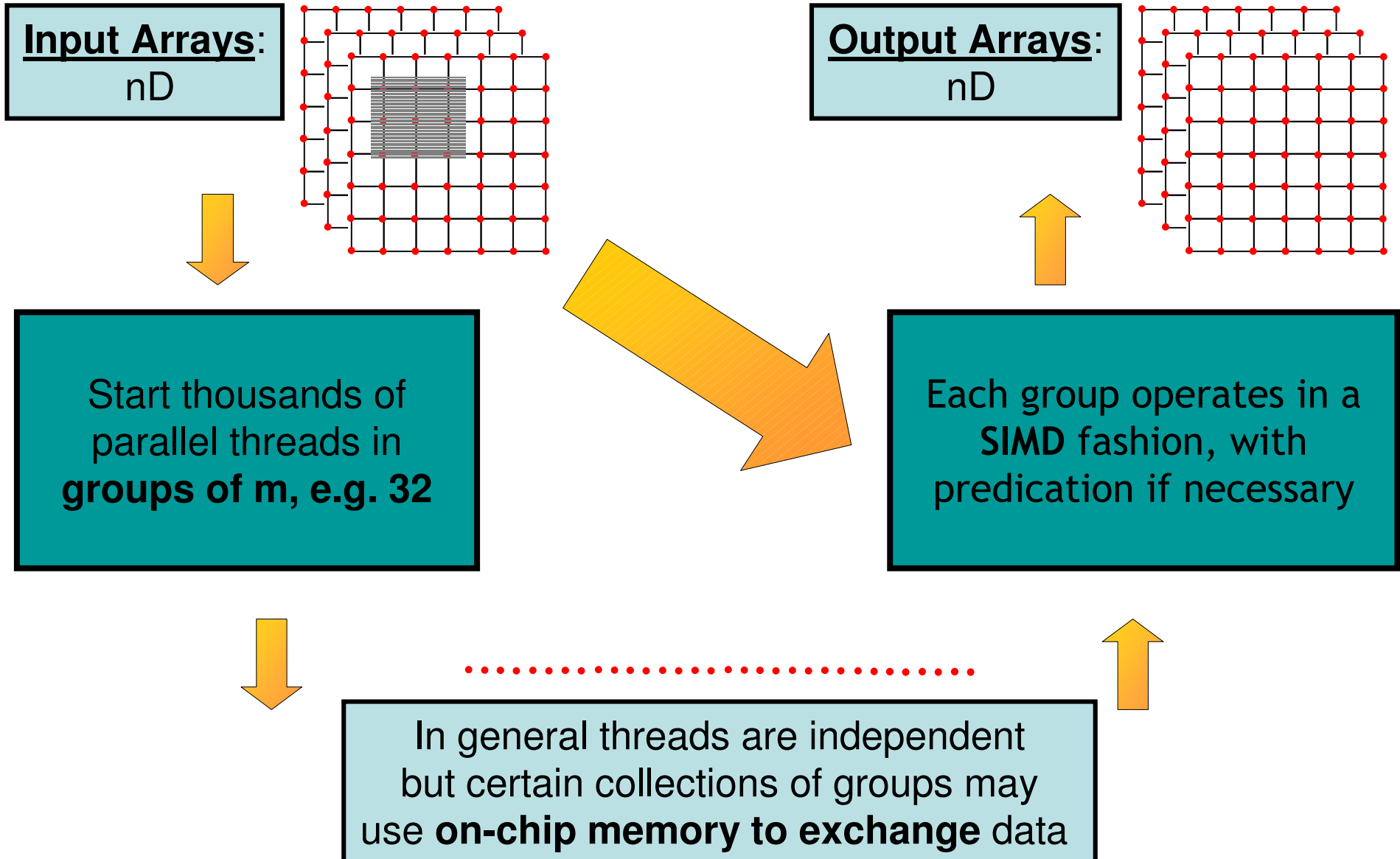
**GPUs often excel in the performance/price ratio**

# Overview

---

- Computation / Bandwidth / Power
- **GPU Characteristics**

# The GPU is a Fast, Highly Multi-Threaded Processor

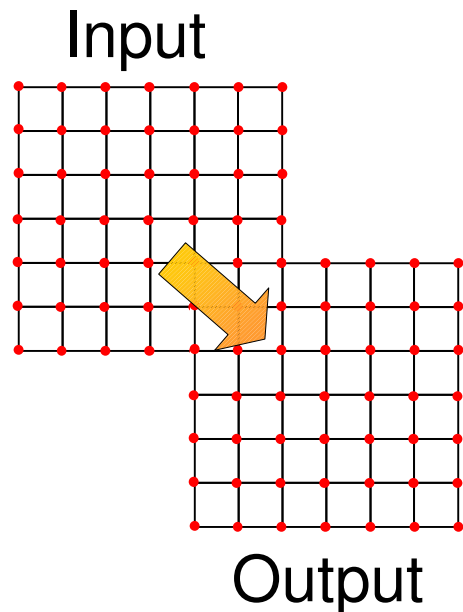


# Input and Output Arrays

---

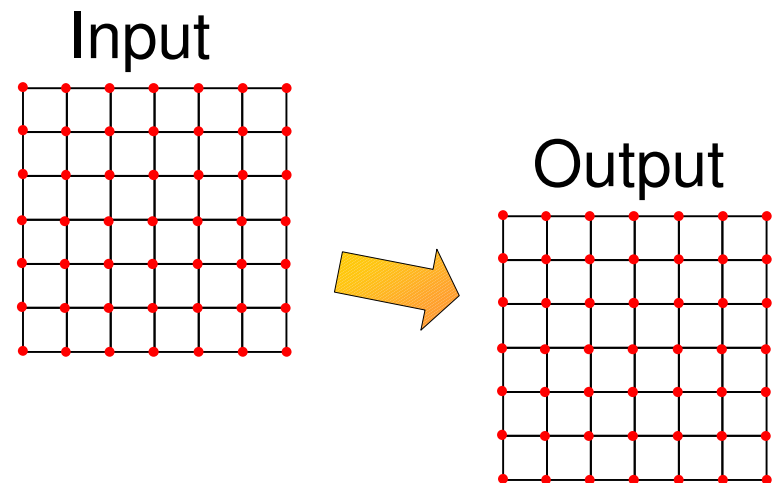
## Single threaded

- Input and output arrays may overlap



## Multi threaded

- Input and output arrays should rather not overlap

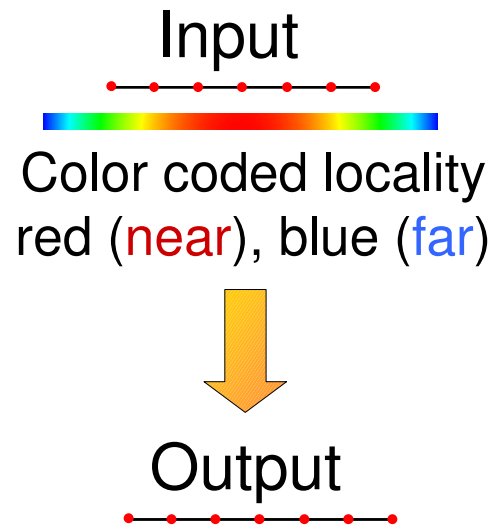


# Native Memory Layout – Data Locality

---

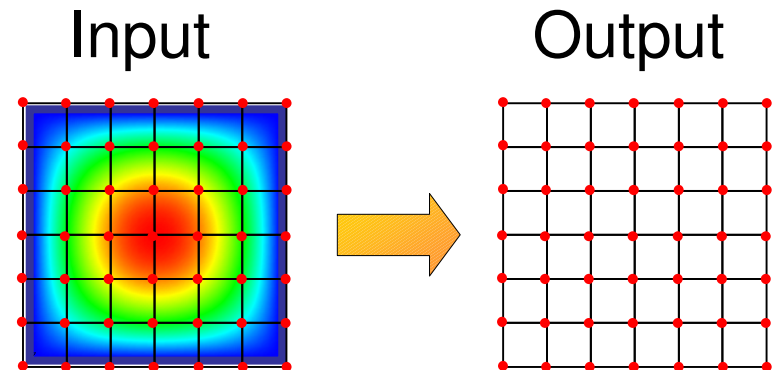
## General memory

- 1D input
- 1D output
- Other dimensions with offsets



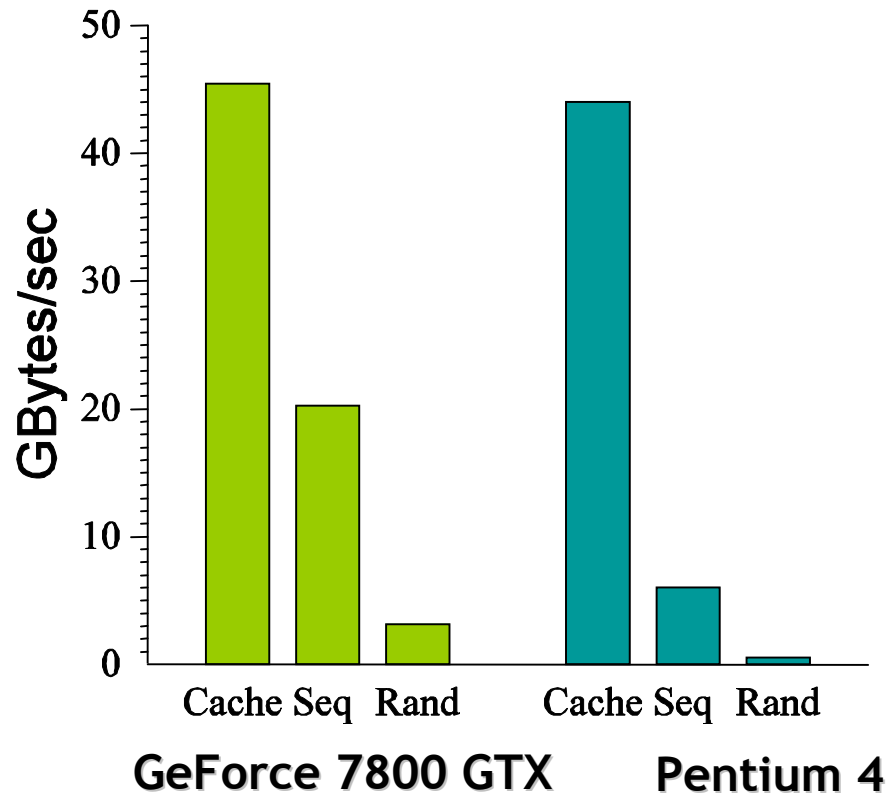
## Texture memory

- 2D input
- 2D output
- Other dimensions with offsets



# GPUs are Optimized for Local Data Access

Memory access types: **Cache**, **Sequential**, **Random**



- CPU
  - **Large** cache
  - **Few** processing elements
  - Optimized for **spatial** and **temporal** data reuse
- GPU
  - **Small** cache
  - **Many** processing elements
  - Optimized for **sequential** (streaming) data access

chart courtesy  
of Ian Buck

# Configuration Overhead

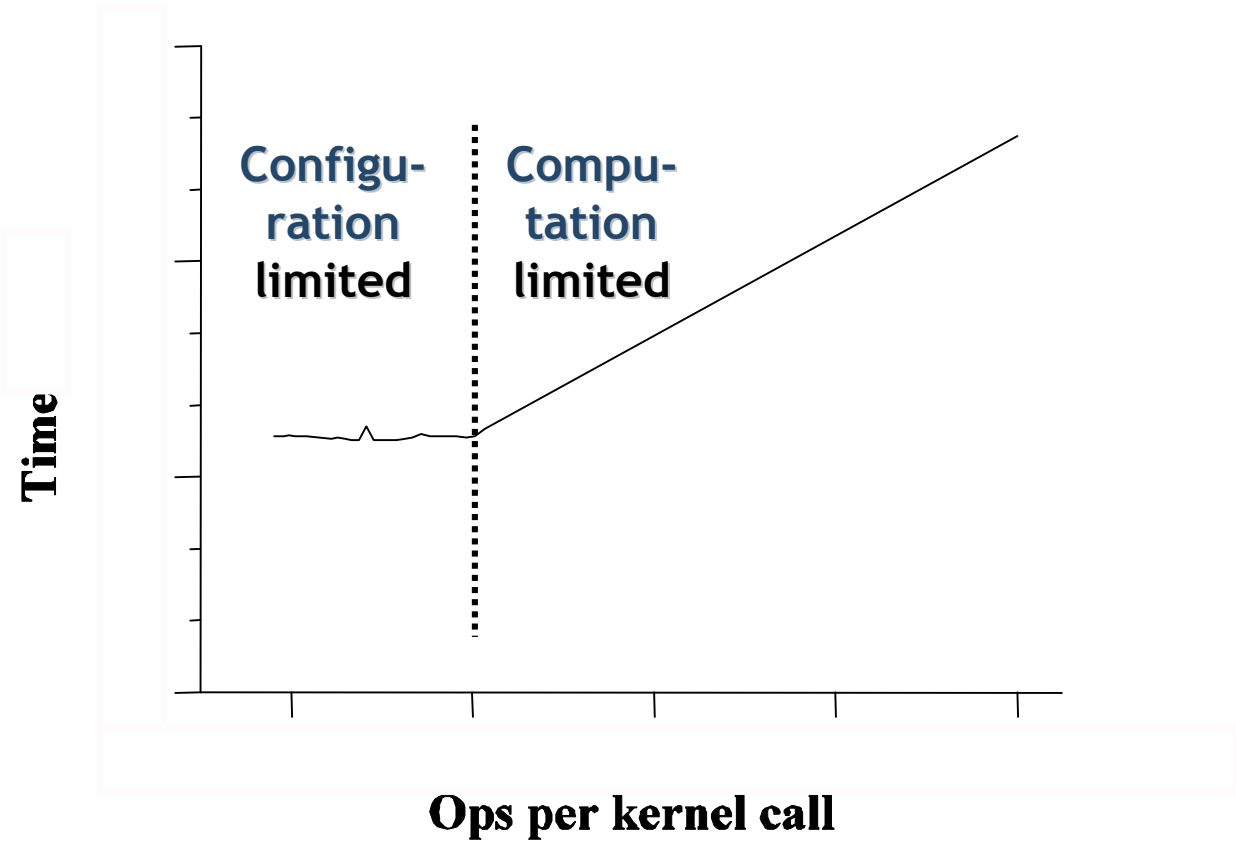
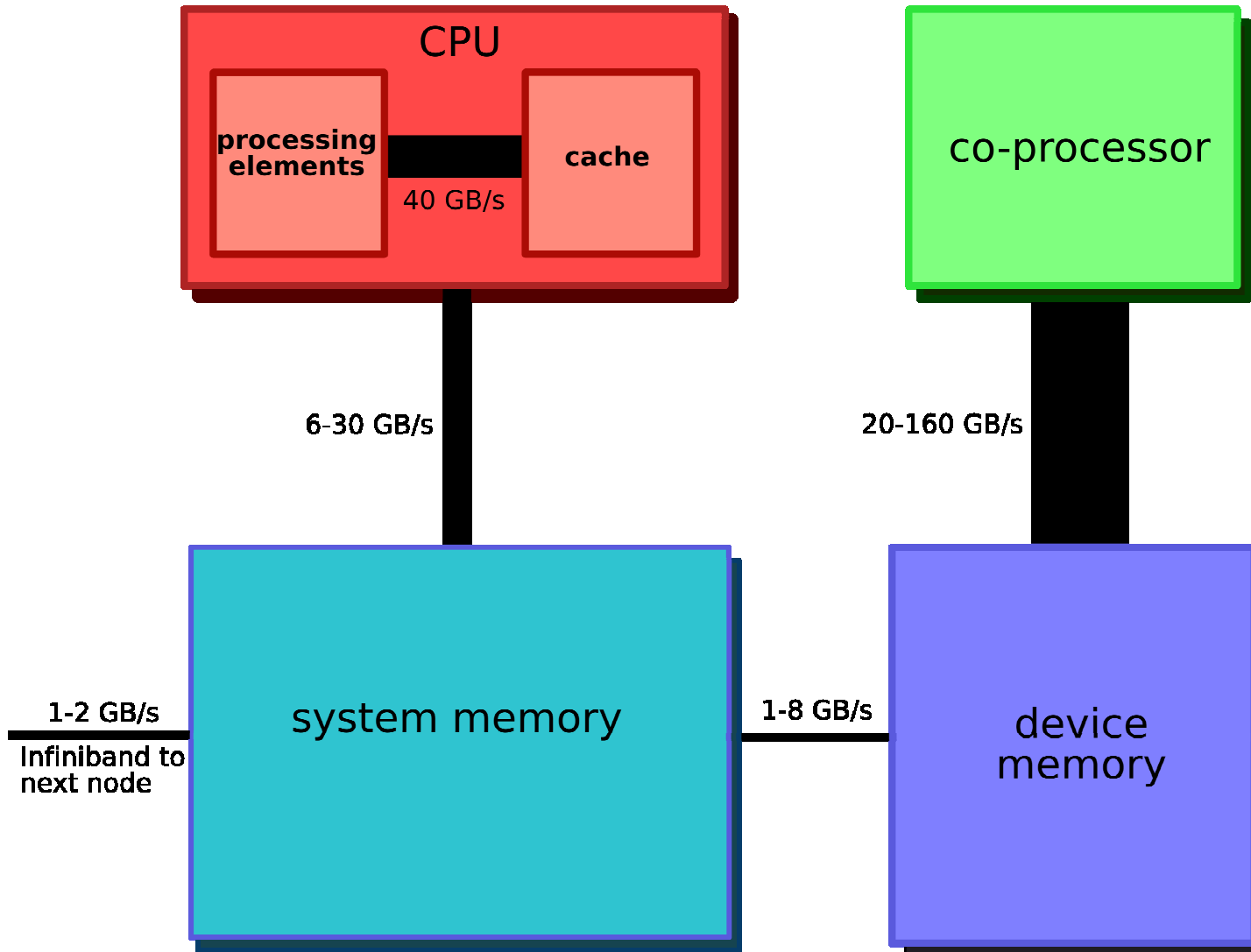


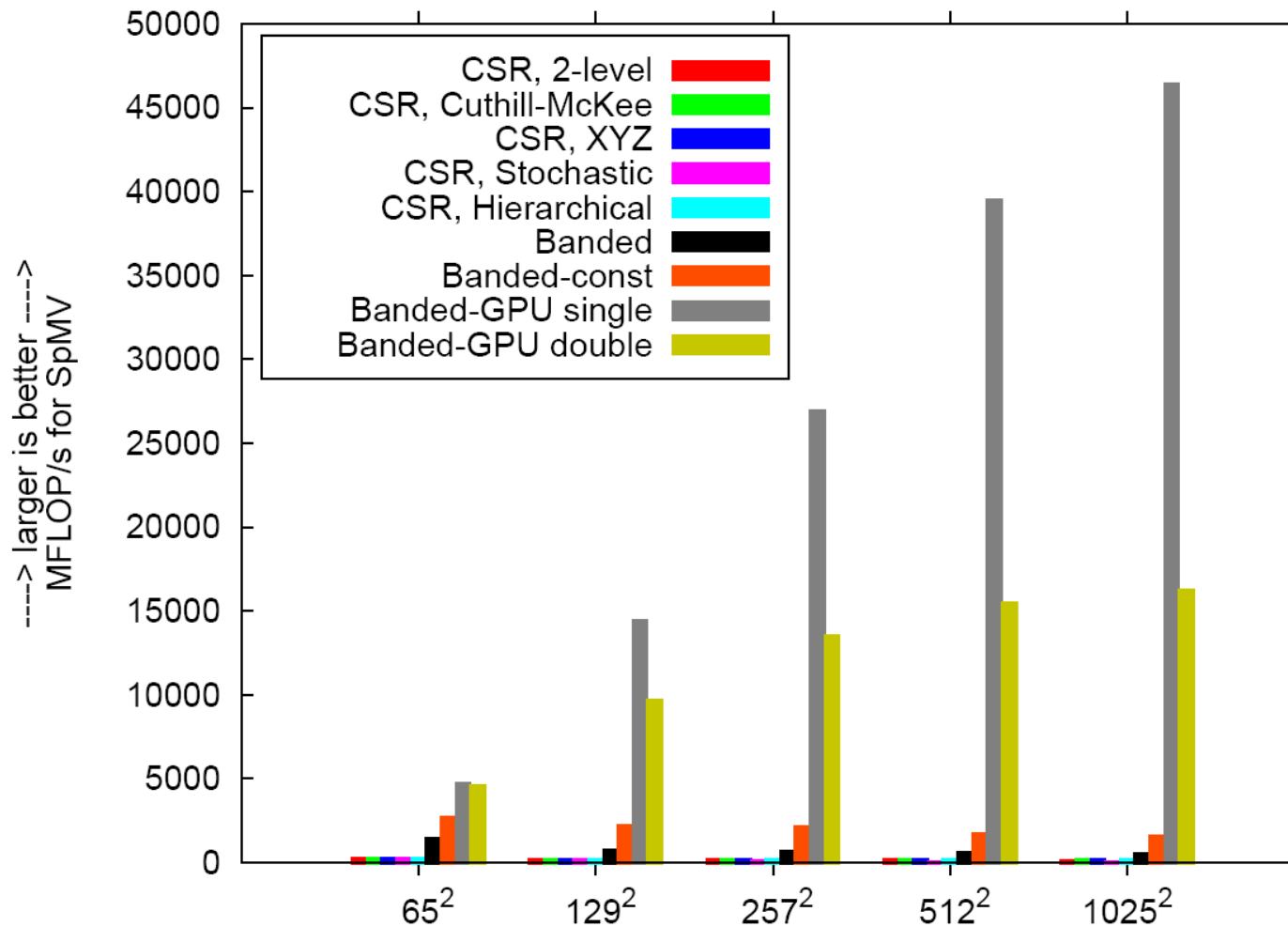
chart courtesy  
of Ian Buck

# Bandwidth in a CPU-GPU System

---



# Sparse MatVec on Tensor Product Grid



- 13GFLOP/s (single) with GPGPU on GeForce 8800 GTX
- 46GFLOP/s (single), 140GB/s with CUDA on GeForce GTX 280

# Conclusions

---

- **Parallelism is now indispensable to further increase performance**
- **For most applications the data – processor locality plays an important role**
- **GPUs offer a fast, inexpensive solution, but understanding the parallel tradeoffs is crucial**