

Assessing the impact of GPU-based computer architecture on PPPL codes (How much should we care about it?)

Stéphane Ethier

Theory Department Research & Review Seminar
Friday, March 7, 2014



NERSC turned 40!!



<https://www.nersc.gov/nersc-40/galleries/historical-systems/>



1974: CDC 6600: The Controlled Thermonuclear Research Computer Center – now known as NERSC – was established in 1974 and unveiled its first supercomputer that same year: a Control Data Corporation 6600 “borrowed” from the weapons program at Lawrence Livermore National Laboratory.

NERSC 1975: CDC 7600



1975: CDC 7600: In 1975, a CDC 7600 replaced the CDC 6600 at NERSC (still known then as the Controlled Thermonuclear Research Computer Center). The 7600 was about 10 times as fast as the CDC 6600 and was capable of performing 20 million mathematical operations per second (or 20 megaflop/s).

NERSC 1978: Cray 1



1978: Cray 1: NERSC acquired a Cray 1 in 1978 and converted the 7600 operating system, utilities and libraries to the new machine, creating the Cray Time Sharing System (CTSS). The Cray 1, which cost about \$8 million (\$25 million in today's money), boasted a million-word memory and was featured in the 1982 Disney movie "Tron."

NERSC 1985: Cray 2



1985: Cray 2: In 1985, the world's first Cray 2 was installed at NERSC (then the Magnetic Fusion Energy Computing Center). It was **the fastest in the world**. Today, an iPad has more computing power...and memory.

NERSC 1992-1996: more Crays...



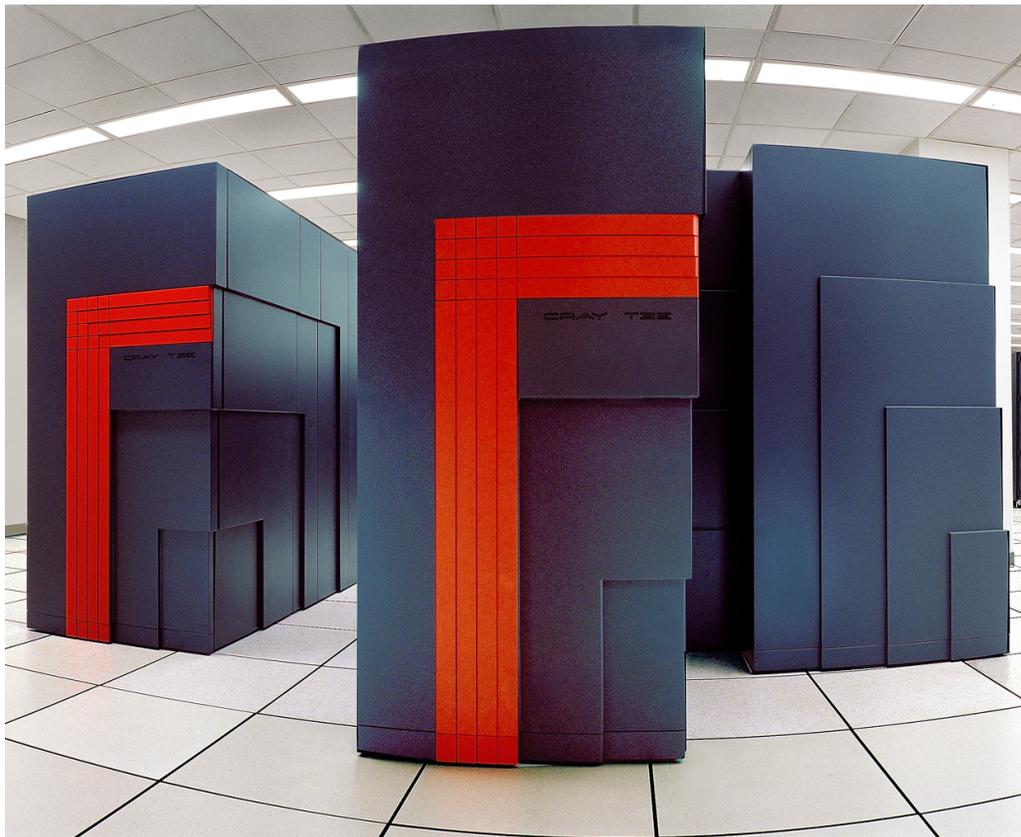
1992: Cray Y-MP C90: NERSC installed Cray's Y-MP C90. The C90 featured a central processor with sustained performance of 1 gigaflop.



1996: Cray J90: NERSC installed its first J90s in 1996 at its future location at Lawrence Berkeley National Laboratory

NERSC 1997: Something new!

FIRST MPP SYSTEM AT NERSC!!



1997: T3E-900 "MCurie": In July 1997, NERSC became the proud owner of a 512-processor Cray T3E-900 supercomputer, which offered 128 gigabytes of memory. Nicknamed "MCurie" after the scientist Marie Curie, NERSC's T3E-900 was ranked No. 5 on the November 1997 TOP500 list of the most powerful computers in the world.

But... who, at PPPL, truly used that machine??



Meanwhile... back at PPPL in 1998

We shared an SGI Origin 2000 computer with the Astro department

Hecate: 64 processors, fully shared-memory, single OS image
but... non-uniform access memory architecture (NUMA)



NERSC 1999: the end of vector systems

Seaborg at its peak: **3,328 cores** (16 cores/node), ~5 Tflops peak
Users forced to use MPI in order to get high performance (charged by node!)
(First GTC simulation of ITER – IAEA 2002: MPI+OpenMP, 1024 processors)



In 1999, NERSC installed an IBM RS/6000 SP supercomputer as its flagship system and named it Seaborg in honor of Berkeley Lab Nobel Laureate Glenn Seaborg. Seaborg was decommissioned in **2007** after providing 25 million CPU hours to users, which resulted in some 7,000 published scientific results.

Not so fast! Remember this guy?

Earth Simulator in Yokohama, Japan



#1 on the top500 list of the fastest computers in the world from 6/2002 to 6/2004. 2 full years!!

**5,120 Vector processors
41 TF peak
3.2 MWatts of power to run it!**



- Wake-up call for the US → “Computenick”
- Revived Cray company by injecting \$\$\$
- Led to the development of Cray X1
- Although Sandia saved Cray with “RedStorm” in 2004

In 2004 DOE establishes the “Leadership Computing Facilities”



OLCF (ORNL)

Computer: JAGUAR
Designed by Sandia (Red Storm project) and built by Cray.

Commercialized by Cray
as the XT3

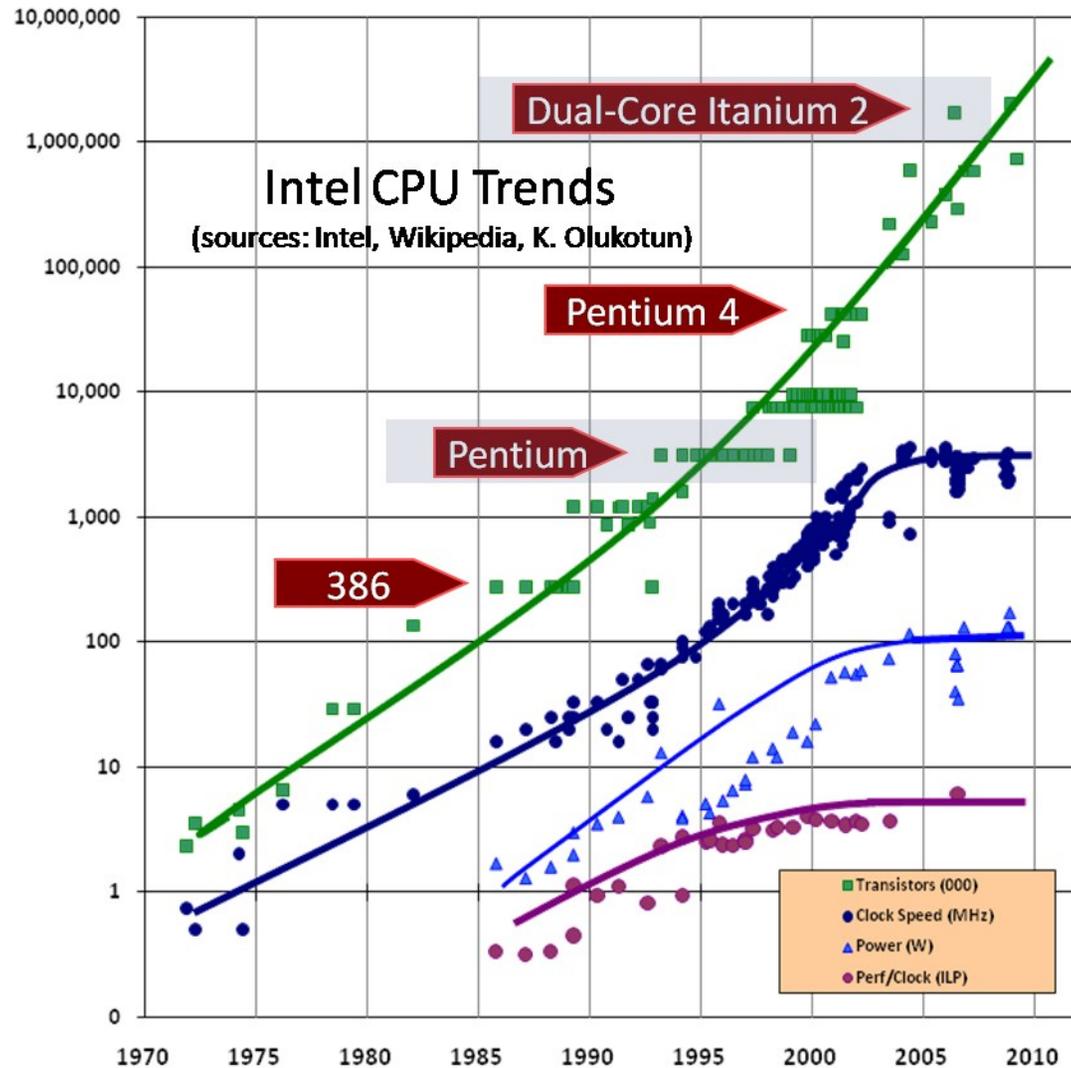
Newer versions are the
XT4, XT5, XE6 ...



ALCF (Argonne National Lab)

Computer: IBM Blue Gene/L (P. Q)
Most energy efficient

Architecture shift in the early 2000s



Heat and power consumption becomes an issue!

Solution:

- Stop increasing frequency
- But CPU won't run faster anymore!

Solution:

multi-core!

NERSC 2007-now: the Crays are back!

All with multi-core processors

Franklin (2007-2012): 38,288 cores,
0.352 PF, 4 cores/processor



Ideal programming model: **MPI+OpenMP**

Edsion (2013-): 133,824 cores,
2.57 PF, 12 cores/proc (24/node)



Hopper (2010-): 153,216 cores,
1.28 PF, 12 cores/processor (24/nodes)



Era of the multi-core processor
is in full swing!!!

BUT WHAT'S NEXT??

OLCF has already made the move to GPU!



TITAN SYSTEM

- 20 PF peak
- CPU+GPU
- #2 on top500

NERSC director Sudip Dosanjh at NUG 2014

NERSC-8 Mission Need



The Department of Energy Office of Science requires an HPC system to support the rapidly increasing computational demands of the entire spectrum of DOE SC computational research.

- Provide a significant increase in computational capabilities, at least 10 times the sustained performance of the Hopper system on a set of representative DOE benchmarks
- Delivery in the 2015/2016 time frame
- Provide high bandwidth access to existing data stored by continuing research projects.
- Platform needs to begin to transition users to more energy-efficient many-core architectures.



What kind of system will NERSC-8 be?

Although architecture for NERSC-8 is not yet known, trend is toward manycore processors



- **Regardless of chip vendor chosen for NERSC-8, users will need to modify applications to achieve performance**
- **Multiple levels of code modification may be necessary**
 - Expose more on-node parallelism in applications
 - Increase application vectorization capabilities
 - For co-processor architectures, locality directives must be added



Manycore?? Is that the same as multi-core?

- No... “manycore” currently refers to GPU-like architecture that has about one order of magnitude more cores per chip than available multi-core processors
 - Examples: Nvidia Tesla GPU (K20, K40), Intel Xeon Phi
- The GOOD NEWS:
 - The way to achieve good performance on these architectures is essentially the same as vectorization on the old Crays!
- The BAD NEWS:
 - The “vectors” need to be much longer in order to keep the hardware busy (= good performance) because memory latency is high
- The secret is to expose as much parallelism as possible from your code and make each item of work independent (no dependencies...)

Using and programming GPUs

- Many possibilities
 - ❑ OpenGL: computer graphics functions used by game developers. **NOT a good idea for scientific codes!!**
 - ❑ CUDA: NVIDIA-specific programming language built as an extension of standard C language. Best approach to get the most out of your NVIDIA GPU. CUDA kernel not portable though so it won't work on the Intel Xeon Phi. Also available for FORTRAN but only for the PGI compiler.
 - ❑ OpenACC compiler directives similar to OpenMP. Portable code. Easy to get started. Available for a few compilers. Not very mature yet but getting better.
 - ❑ Libraries, commercial software, domain-specific environments, . . .
 - ❑ OpenCL: open standard, platform- and vendor independent
 - Works on both GPU AND CPU!!
 - Even harder than CUDA though...

The PPPL GPU effort...

- XGC1 has already made the jump!
 - ❑ Ported to GPU using PGI CUDA FORTRAN
 - ❑ Runs production simulations on OLCF Titan computer, a CPU-GPU system currently #2 on the top500 list
 - ❑ CPPG Dr. Jianying Lang is GPU expert assigned to this project
 - ❑ Help from ESPI project collaborators from ORNL
- GTS
 - ❑ Collaboration with NVIDIA engineer
 - ❑ Making progress but not in production yet
- GTC-P (PPPL/LBNL collaboration)
- ESC-EEC (Zakharov)
 - ❑ Particle orbit part was ported to GPU in ~2 days by Xujing Li!
- ORBIT
 - ❑ NUF summer student Ante Qu from Princeton University made an OpenMP, OpenACC, and CUDA Fortran version of ORBIT → **56X faster**
 - ❑ Only the core of the code was modified

So the answer to the original question is...

- **YES!** We should care about porting PPPL codes to the manycore architectures in order to achieve good performance in the future. Without changes, these codes could end up running slower!
- Lots of codes to work on:
 - ❑ The flagship codes: M3D-C1, GTS, XGC1 (Xeon Phi)
 - ❑ GTC-NEO, M3D-K, HYM, SPEC, GKEYLL, ... (sorry to those I'm missing...)
 - ❑ I believe that TRANSP could benefit as well
 - ❑ Some work was done for PIES a few summers ago
 - ❑ Space physics codes
 - ❑ Experimental codes? SPIRAL?
 - ❑ We need to build the expertise.

Recommendation

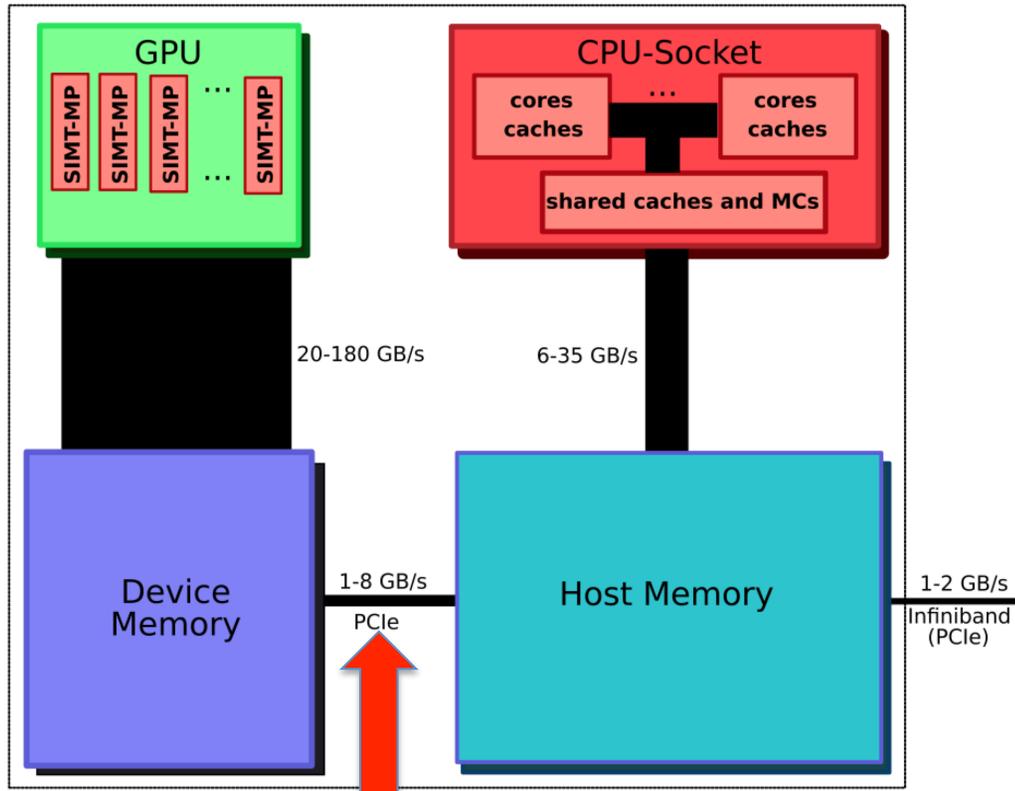
- Still lots of room for improvement on current CPU architecture
- Start with OpenMP then move to OpenACC 2.0
- Use CUDA for those who want max performance

Side-by-side comparison of CPU vs. GPU

Processor	Intel Xeon E5-2690 “Sandy Bridge”	NVIDIA K20 Tesla GPU “Kepler” (Fermi)
No. of cores per “node”	16 cores (2x 8/chip)	2,880 (512)
Frequency	2.9 GHz	~1 GHz
Memory bandwidth	51.2 GB/sec	~320 GB/s (177)
Peak Flops	371 GFlops	~2000 Gflops (665)
Memory (shared)	32 – 64 GB	6 GB

- The Kepler K20 will be the GPU in the Titan system at ORNL
- The Cray Cascade system “Edison” at NERSC has the latest version of the Intel Ivy Bridge processor

Current model: a CPU is required to drive the GPU



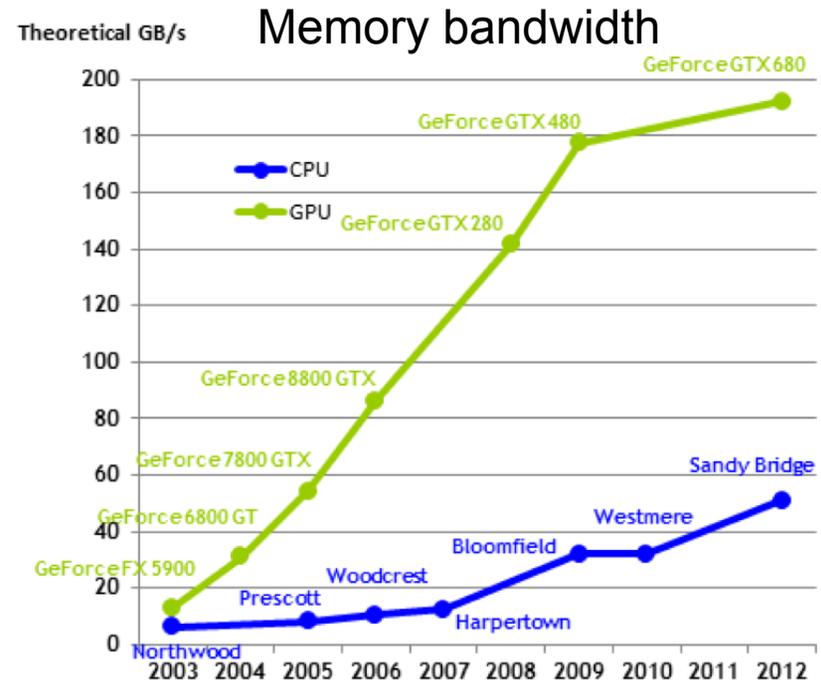
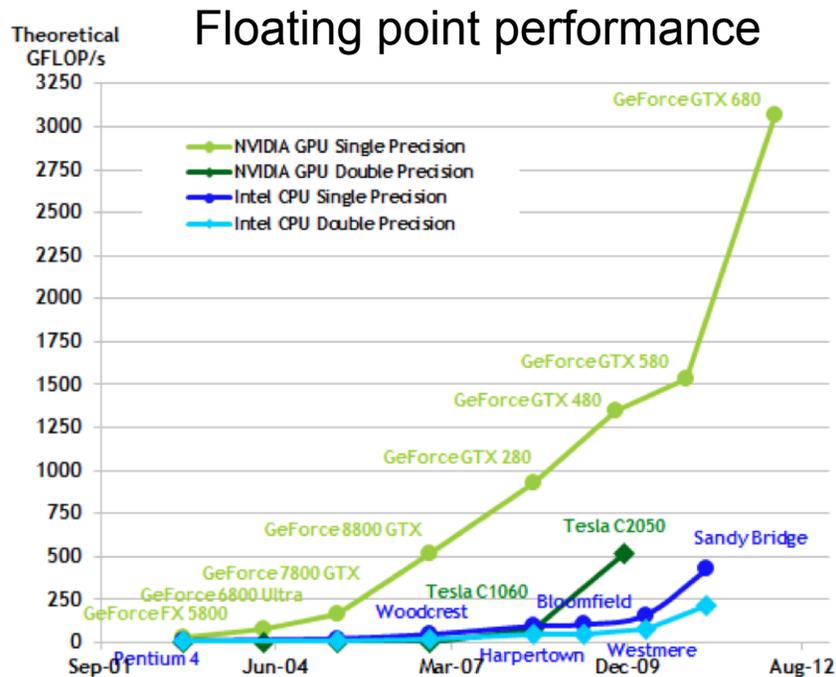
Bottleneck in many cases

Ideally, both GPU and CPU should be working concurrently

This is temporary!

The architecture is moving toward integration

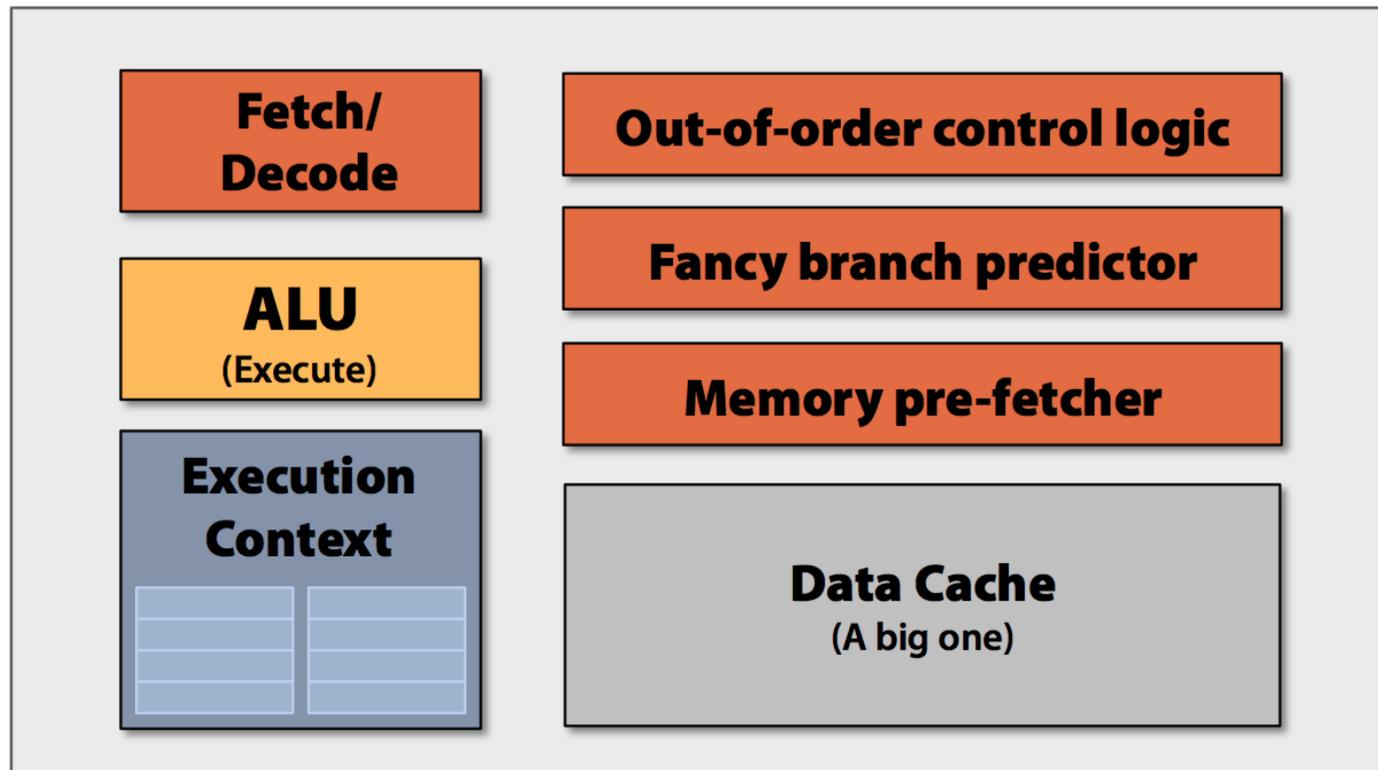
The standard performance plots comparing GPU and CPU



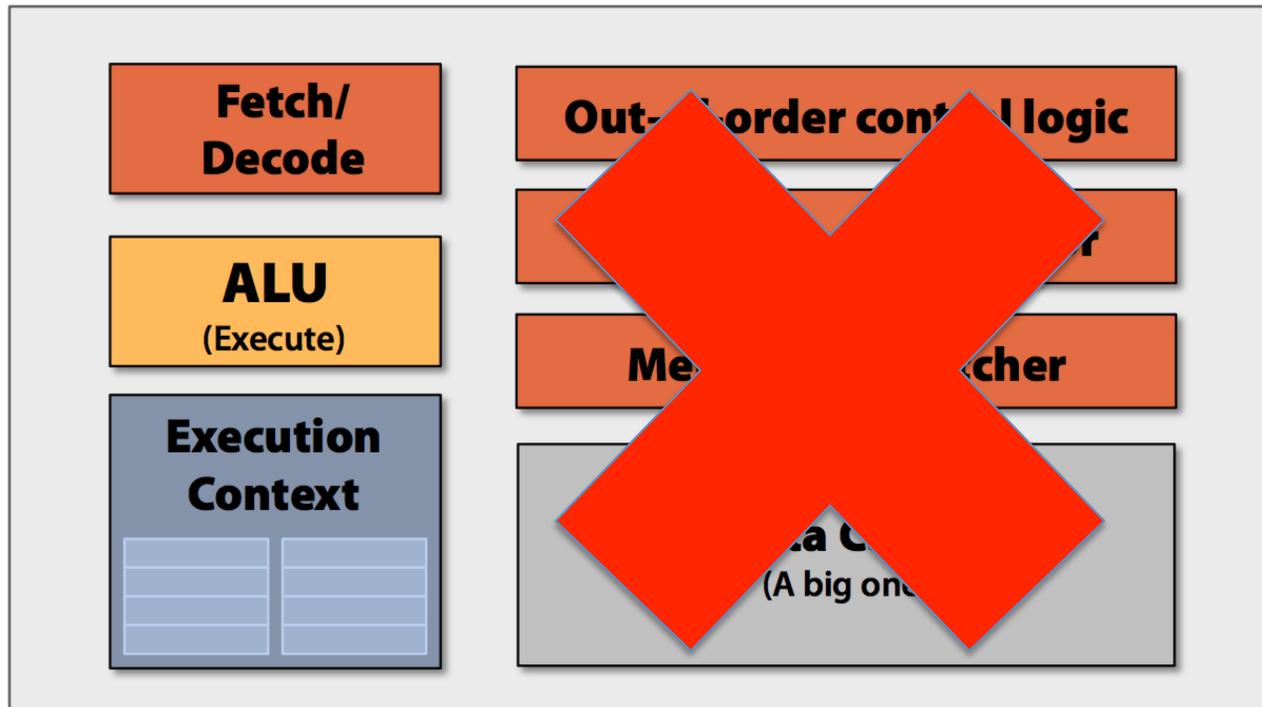
*Plots from CUDA C Programming Guide Version 4.2



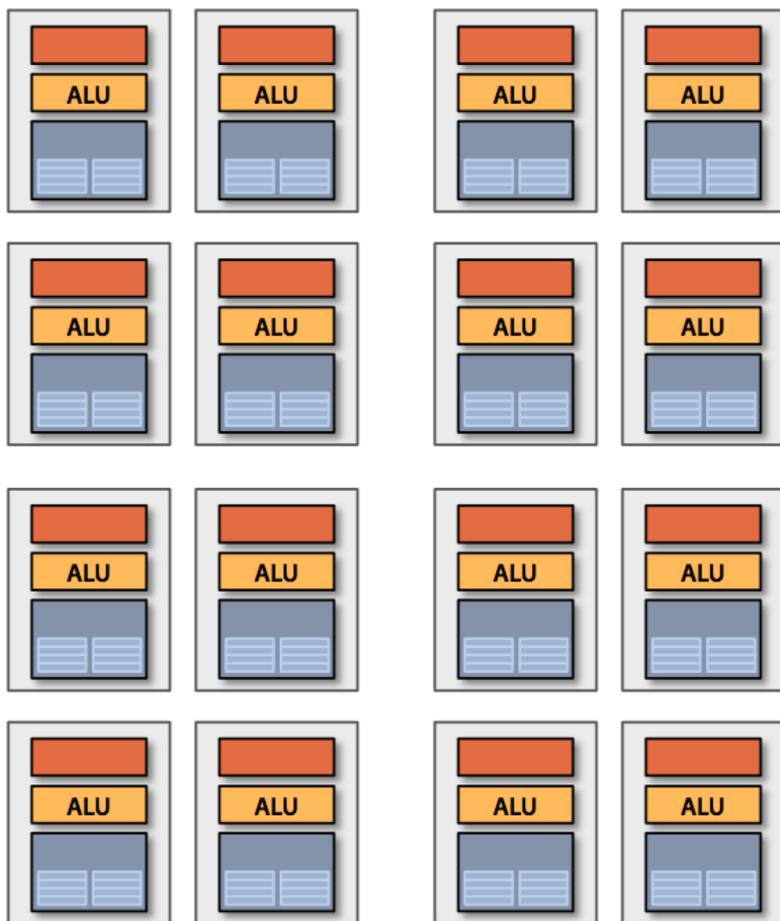
Start with a “CPU-style” core...



Remove everything that makes a single instruction stream go fast



Put many of these simple cores together



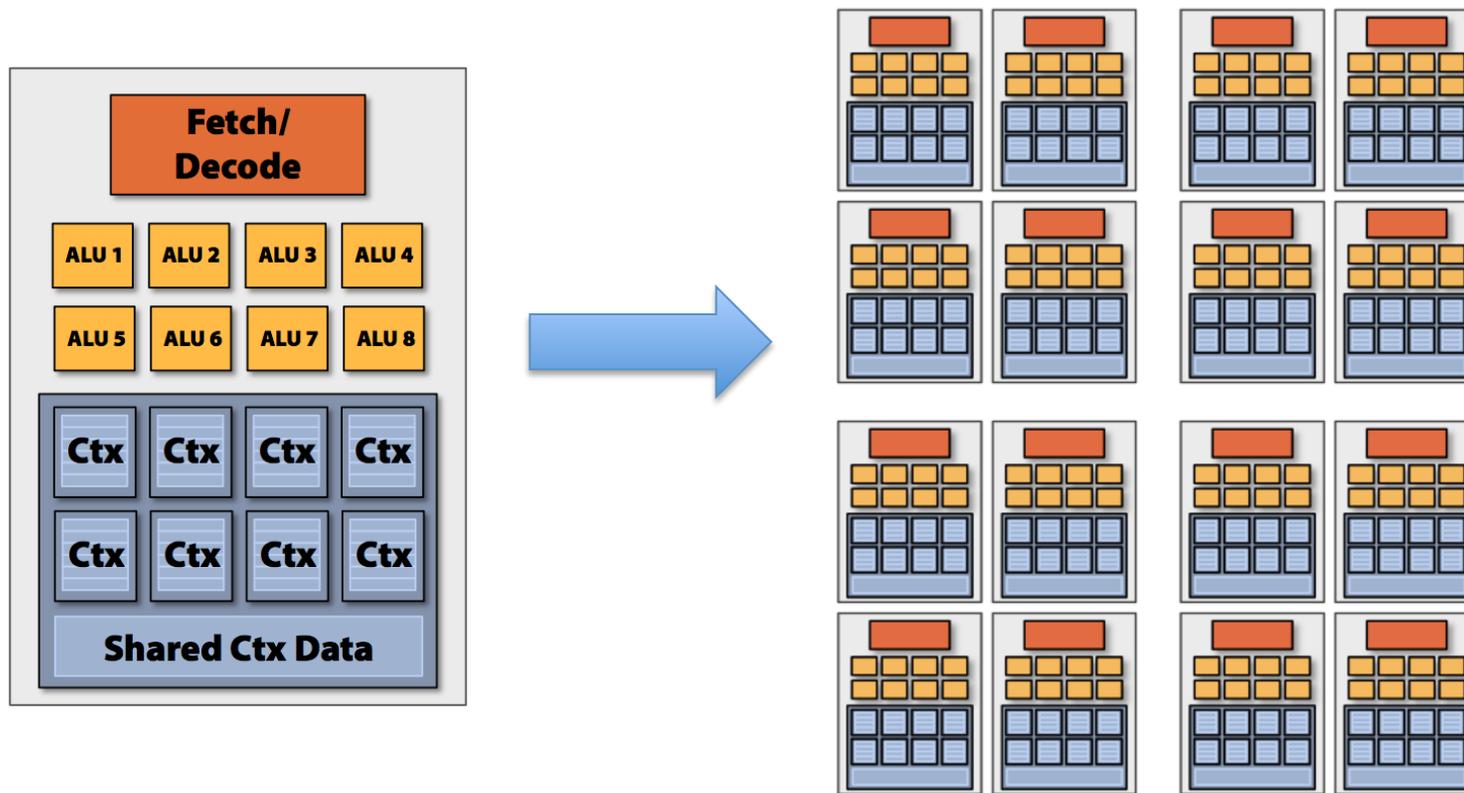
- Let's not forget the original purpose of the GPU though:
 - Same operations applied to a large number of vertices, pixels, polygons, ...

= DATA PARALLEL or SIMD (Single Instruction Multiple Data)

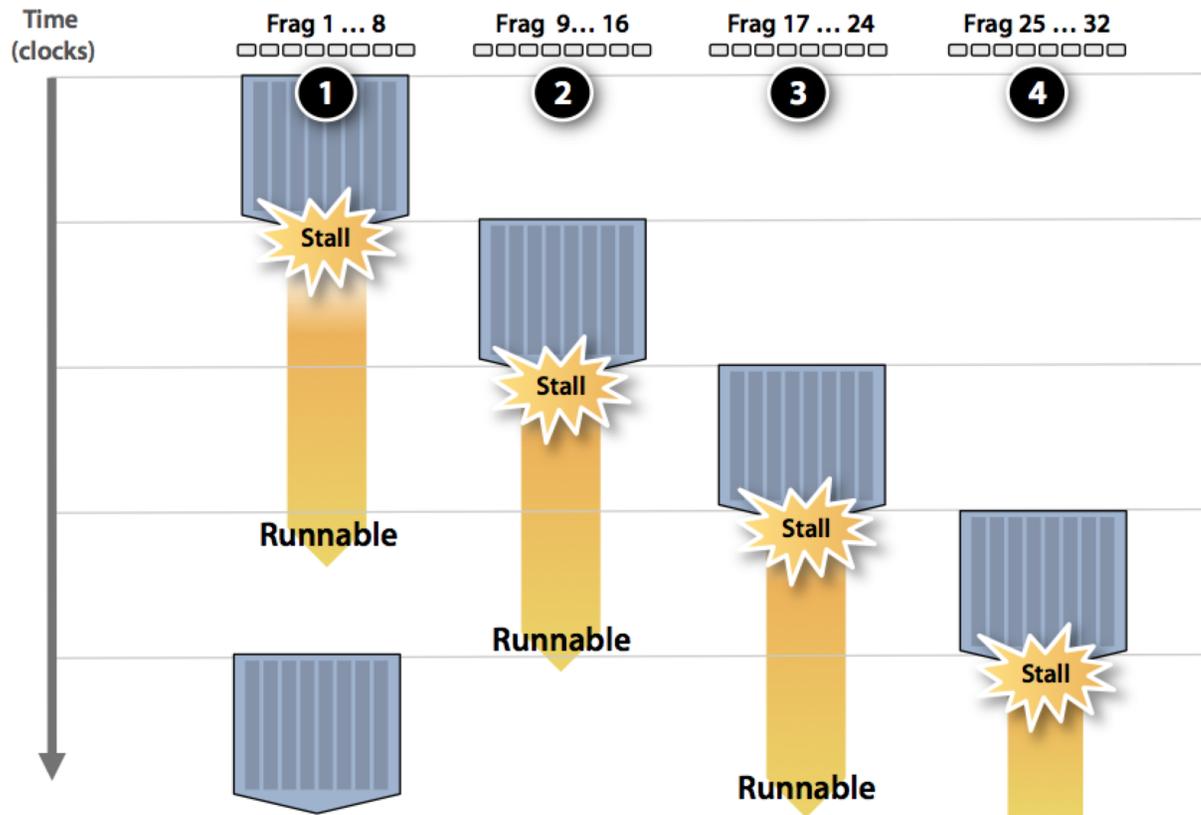
Better known in our community as

VECTOR PROCESSING

So let's add some arithmetic units and have them share a stream of instructions



The secret to GPU high throughput: massive multi-threading + interleaving

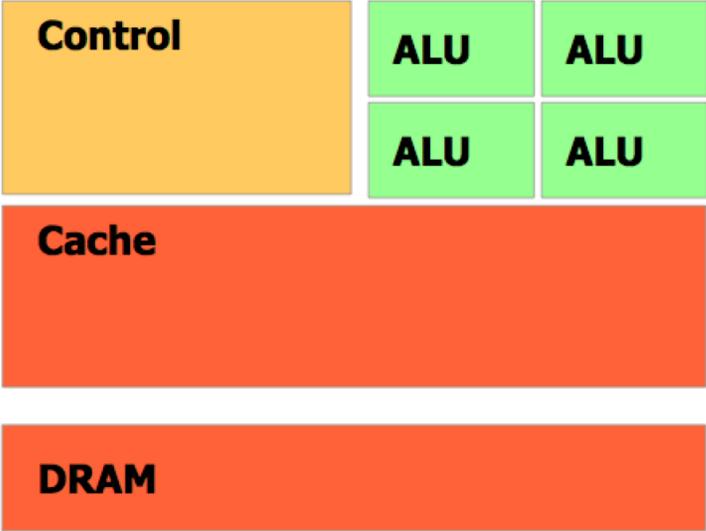


NOT SIMD
But rather

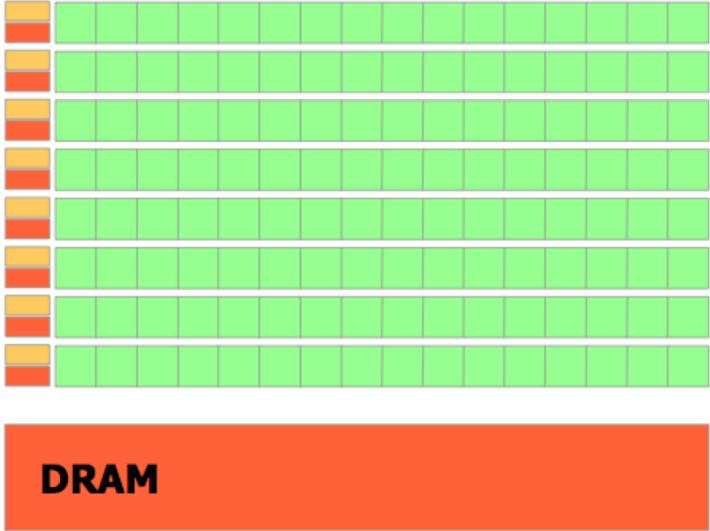
SIMT!
Single
Instruction
Multiple
Threads

**255 registers
per thread!!**

Classic picture of CPU vs. GPU



CPU



GPU

What to do first...

- MOST IMPORTANT:
 - ❑ Find and expose as much parallelism as you can in your code.
 - ❑ Try to remove as many dependencies as you can between successive iterations in a loop.
 - ❑ The ideal case is when each iteration is completely independent from the others → VECTORIZATION

Try OpenACC directives first

- <http://www.openacc-standard.org>
- http://www.pgroup.com/doc/openACC_gs.pdf
- Least changes to your code
- It is portable across different platforms and compilers
- Not all compilers support Open ACC though
 - ❑ CRAY, PGI, and CAPS are the only ones at this point
- When running the same code on a multi-core CPU you can use OpenMP directives instead of OpenACC and run in parallel there too!
- Hides a lot of the complexity
- Works for Fortran, C, C++

Example of OpenACC directive

It can be as simple as the following:

```
subroutine smooth( a, b, w0, w1, w2, n, m, niters )
  real, dimension(:, :) :: a, b
  real :: w0, w1, w2
  integer :: n, m, niters
  integer :: i, j, iter
  do iter = 1, niters
    !$acc kernels loop
    do i = 2, n-1
      do j = 2, m-1
        a(i, j) = w0 * b(i, j) + &
          w1 * (b(i-1, j) + b(i, j-1) + b(i+1, j) + b(i, j+1)) + &
          w2 * (b(i-1, j-1) + b(i-1, j+1) + b(i+1, j-1) + b(i+1, j+1))
      enddo
    enddo
  enddo
```

OpenACC not giving good performance? move to CUDA

- CUDA C
 - http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
- CUDA FORTRAN
 - <http://www.pgroup.com/doc/pgicudaforug.pdf>

Some GPGPU references

- <http://www.gputechconf.com/gtcnew/on-demand-gtc.php>
- <http://www.nvidia.com>
- <http://gpgpu.org>
 - In particular: <http://gpgpu.org/ppam2011>
- <http://www.olcf.ornl.gov/event/cray-technical-workshop-on-xk6-programming/>
- <http://www.pgroup.com/resources/index.htm>
- <http://www.caps-entreprise.com/products/openacc-compiler/>

Conclusion

- No matter where you will run your code tomorrow, you need to exploit all the parallelism and think in terms of shared memory multi-threading
- This is valid for both CPU and GPU
- In the future, CPU and GPU will merge to become a highly power efficient, highly multi-threaded compute hardware