Hardware/Software Divergence in Accelerator Computing

Volodymyr Kindratenko
Innovative Systems Laboratory

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

How do you envision the role of accelerators in parallel computing and high performance computing in the next decade including the role in the exascale systems?

- As means to claim the highest peak performance, but not as means to achieve the highest efficiency at scale ☺
  - Tianhe-1A was #1 on Top-500 in 2010
  - Its Rmax was about 54% of its Rpeak
- Accelerators will play a minimal role in extreme-scale systems
  - Sure, systems such as Titan and BW will have a lot of them
  - But we have yet to see what performance the application scientists will achieve on these systems using GPUs **at scale**
- Accelerators will play a substantial role for small-scale systems
  - A system with O(10) of GPUs can replace a cluster with O(1000) CPU cores for applications that can sustain strict scaling limitations imposed due to accelerators

How do you view the hardware/software divergence in accelerator Computing?

- It is getting worse
  - We know how to build O(1M) CPU cores systems
  - But we do not know how to write software that can take advantage of such systems
- Accelerators add another layer of complexity to already overly complex systems
  - Heterogeneity in hardware also means greater degree of divergence in software: host code, accelerator code, communication layer, etc.

Which accelerator (hardware) do you think will have advantages in the next 10 years and most likely win the battle in the next decade and why?

- Intel Many Integrated Core (MIC) Architecture –like accelerators will eventually win the battle.
  - The architecture is sound (many cores, wide vector units, high memory bandwidth)
  - Programming model is very flexible, ranging from kernel offload co-processor to running entire application on the MIC
  - Programming tools are conventional: icc, idb, vtune
  - Programing languages are familiar: C/C++ with pragmas and libraries
  - Software development effort on MIC is comparable with performance tuning effort rather than with code reimplementation
- Oh yes, when the “war” is over, what we consider today to be an accelerator, will be in our mainstream processor

- Why not NVIDIA GPUs?
  - Market forces are working against NVIDIA
    - With the introduction of APUs, Intel and AMD are taking away the low-end discrete GPU market from NVIDIA
    - Without this low-end mass-market, NVIDIA will have a harder time justifying the expense of developing high-end GPUs
    - Market for high-end (HPC) GPUs is too small to sustain NRC
  - Software development efforts necessary to efficiently utilize GPUs are substantial, despite all the efforts by NVIDIA and its partners developing tools and compilers
  - Programming model (kernel offload co-processor) is inherently limited
  - NVIDIA CUDA SDK is great, but it locks the developers into a particular (*incompatible with the rest of the world*) software-hardware environment
    - Other approaches, such as OpenCL, have yet to deliver performance levels achievable with CUDA

What programming model/library of accelerated computing do you think will have advantages in the next 10 years and most likely win the battle in the next decade and why?

- Anything that is easy to use without sacrificing performance
  - Libraries for applications which heavily rely on standard libs (fft, linear algebra, …)
  - Kernel offload for codes with distinct, well-defined and dense computational kernels

What research challenges do you envision will be most critical and should be addressed in the coming years for the success of accelerator computing?

- Ease of use
- Programmer's productivity
- Automation (auto parallelization, auto-vectorization, auto-tuning, …)

- Communication bottleneck