Answers to the Questions for NSF XPS Workshop 2015

Mahmut Taylan Kandemir (Penn State)

1. Applications: What are the applications that motivate future systems? What advances in algorithms and programming systems will support these applications?

I believe applications that are composed of “learning engines” will dominate the future computer landscape, ranging from embedded devices running Android to high-performance computing (HPC) systems. Since the learning activity itself is “approximate”, I would imagine that “tunable approximation” (which will allow programmers to trade off accuracy with performance/power benefits) will be a key component that needs to be supported by both algorithms and programming communities. It is also possible that most of future applications (even mobile ones) will actually run in a server and users will access mostly the front-ends of these applications.

2. Systems: What hardware architectures and distributed systems will support future applications? What challenges do we face in design and management?

Supporting approximate computing in a scalable fashion will demand cooperation from multiple layers, including application, programming languages, compilers, OS, runtime system and hardware. For example, hardware needs to employ multiple units, each capable of performing the same function, but with a different accuracy-performance/power tradeoff. Alternately, the same unit can be reconfigured to operate at a different accuracy-performance/power tradeoff points in different parts of the execution, depending on user and/or execution environment needs. Since future applications will be even more data intensive than what we have today, provisioning near-data computing capabilities in hardware (e.g., performing computations in cache controllers) will be critical. Further, with increasing dataset sizes, cache hierarchies may be less effective, and optimizing for miss latency could be more important than optimizing for hit rate.

3. Technologies: What emerging technologies will change fundamental assumptions in hardware and software? What constraints disappear? What challenges arise?

Growing employment of nonvolatile technology (e.g., storage-class memory) brings both opportunities and problems. Some of these technologies also present a tradeoff between durability and performance/power, which can have an impact on software design and deployment. Also, if successful, using optical communication within a chip can open up new venues for both hardware and software design. In particular, with broadcasting being a viable possibility, software can be restructured based on availability of data in the neighborhood, instead of the original order of computations.
4. Methodologies: How should we perform interdisciplinary research that spans applications, systems, and technologies? How should abstraction layers evolve?

One option is to set up an association that includes leading companies (chip manufacturers and software developers), academia and national labs, with the goal of defining and promoting open specifications that will guide the development of parallel hardware and software. These specifications can primarily capture the interfaces between different layers and can evolve over time.

5. Risks: What are the risks that threaten the success of XPS research directions? How do we guard and hedge against these threats?

Two critical threads may be the lack of well-articulated strategies to express parallelism and locality properties/needs of applications, and the mismatch between static and dynamic parallelism. For the first one, it has been proven over the years that, going beyond the conventional thread/task concept is difficult, yet many applications require alternate strategies to express available (intrinsic) parallelism at different scales. It is not clear whether adopting, say, a different programming construct for each level of parallelism is a viable option. For the second one, most of the time, the compile-time parallelism may not fit well to runtime opportunities/possibilities for parallelism, and bridging this gap (e.g., by preparing for different runtime scenarios at compile time or employing a sophisticated runtime system which is capable of low-cost adaptation) will be quite challenging.