(a) Applications: What are the applications that motivate future systems? What advances in algorithms and programming systems will support these applications?

Data-parallel applications involve many similar tasks operating on distinct data. This class of applications are found in a wide variety of demanding application domains, including graphics rendering, computer vision, audio/video processing, physical simulation, machine learning, data mining/analytics, and graph processing. There is no reason to suspect this will change in the future. Data-parallel applications will continue to be critical workloads across the computing spectrum: from tiny embedded devices, to the data center, to super computers. What is changing is the characteristics of these data-parallel applications. *Traditional data-parallel applications* have a large number of similar tasks that contain regular control flow, operate on regular data structures with disjoint inputs and outputs, and use static task generation. Emerging *amorphous data-parallel applications* also have a large number of similar tasks, but the control flow in these tasks is more irregular, the tasks operate on irregular data structures with sometimes overlapping inputs and outputs, and the tasks can dynamically generate new tasks based on the input data. Amorphous data-parallel applications are significantly more challenging than traditional data-parallel applications to efficiently map to data-parallel accelerators such as general-purpose graphics processing units, short-vector SIMD extensions, and more specialized data-parallel accelerators. There is a need for interdisciplinary research on new programming frameworks, programming languages, compilers, and runtimes to support efficiently executing emerging amorphous data-parallel applications on future data-parallel accelerators.

Another important trend is the growing popularity of dynamic programming languages from smartphones (e.g., JavaScript for mobile web clients), to servers (e.g., Node.js, Ruby on Rails), to supercomputers (e.g., Julia for technical computing). Dynamic programming languages such as Python, JavaScript, PHP, Ruby, and MATLAB are all now among the top-ten most popular programming languages. These *productivity-level languages* (PLLs) leverage lightweight syntax, dynamic typing, reflection, and extensive standard libraries to enable agile software development methodologies. However, simple interpreters for PLLs are often orders-of-magnitude slower than statically compiled languages such as C/C++. There is a continuing need for interdisciplinary research on software and hardware techniques to close this *productivity-performance gap*.

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

It is becoming clear that computer system designers are increasingly turning to a heterogeneous mix of general-purpose multicores, programmable data-parallel accelerators, reprogrammable logic, and specialized accelerators integrated onto a single die. Unfortunately, these heterogeneous systems give rise to a host of new software and hardware challenges. Programmers must choose between general-purpose parallel programming frameworks for multicores (e.g., POSIX threads, Cilk++, Intel TBB), many different data-parallel programming frameworks for different data-parallel accelerators (e.g., Cuda, OpenCL), hardware description languages (e.g., SystemVerilog, VHDL, Bluespec, Chisel, PyMTL), and custom frameworks for specialized accelerators. Separate compilers and runtimes are required for the different types of engines, and integrating multiple heterogeneous engines requires completely different hardware design teams. Finally, depending on the application characteristics
usually one type of engine is under-utilized while the other limits performance. This heterogeneity clearly increases complexity and costs at all levels of the computing stack. There is a need for interdisciplinary research that explicitly focuses on mitigating this software and hardware design complexity while maintaining the critical benefits of heterogeneity.

The context for amorphous data-parallel applications, applications written in dynamic programming languages, and heterogeneous architectures is increasingly moving to the extremes of the computing spectrum: small-scale, highly constrained embedded devices which will make up the future internet-of-things, and large-scale, highly distributed cloud/supercomputing platforms. At the small-scale end of the spectrum, heterogeneity is critical to enabling sophisticated information processing within strict energy constraints, while at the large-scale we are already seeing a trend towards integrating general-purpose graphics processing units into top-tier supercomputers (e.g., 15% of the TOP500 use accelerator/co-processors with NVIDIA GPGPUs or Intel Xeon Phis), and integrating field-programmable gate-arrays into data-centers (e.g., recent work on Catapult by Microsoft).

(c) Technologies: What emerging technologies will change fundamental assumptions in hardware and software? What constraints disappear? What challenges arise?

The inevitable end of technology scaling will radically change fundamental assumptions in hardware and software. There are simply no compelling technologies on the horizon that will offer a “silver-bullet” and thus continued exponential technology scaling. Such emerging technologies as memristors, spintronics, quantum computing, and nanophotonics are all promising, but they are far from being reliable and realistic replacements for the current state-of-the-art. Hardware specialization appears to be the most promising approach that is realizable with current technologies, so the end of Moore’s law could usher in a new “golden era” of hardware design. While it is of course important to explore cross-layer research on emerging technologies, one might argue that it is even more important to focus on cross-layer approaches to address the gap between the end of technology scaling and the (potential) adoption of an emerging technology.

(d) Methodologies: How should we perform interdisciplinary research that spans applications, systems, and technologies? How should abstraction layers evolve?

One key to interdisciplinary research that spans applications, systems, and technologies is an increased emphasis on unified research frameworks. Currently, application-level research usually relies on available hardware platforms since simulators of emerging architectures are simply too slow; architecture-level research often focuses on cycle-approximate simulation with rough analytical models for first-order area, energy, timing estimation; VLSI-level research focuses on cycle-accurate hardware implementation with limited design-space exploration. Since one’s research is only as good as one’s tools, unified frameworks that span applications, systems, and VLSI are the key to credible interdisciplinary research. There is a continued need for funding work on such new unified frameworks.

Another key to interdisciplinary research is the open distribution of research artifacts such as application code, datasets, simulators, and VLSI implementations. When these artifacts are made public, it is easier for researchers in different communities to leverage these artifacts to make important advancements. There is an urgent need for our communities to recognize the importance of open distribution of artifacts, and to continue discussion on how to best incentivize and facilitate such distribution.
(e) Risks: What are the risks that threaten the success of XPS research directions? How do we guard and hedge against these threats?

One of the key distinguishing features of the NSF XPS program is its requirement for an interdisciplinary team of PIs. If taken seriously, this requirement forces PIs to push outside their comfort zone and to engage other research communities. There is a risk that this requirement may be weakened or even eliminated. I would argue for strengthening this requirement to ensure that any funded XPS project credibly incorporates two PIs from truly distinct research communities.