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

All the applications that have been neglected thus far by the parallelism fairy – unglamorous software that runs on consumer machines, which should get faster with more cores but doesn’t: web browsers, file parsers and loaders, batch operations like PDF conversions. Also there are many well-understood categories, like sorting, where parallel sorts are available but most programmers don’t use them. Rather they use whatever is built into their language or programming environment. How do we make parallelism safe and predictable enough that it is “on by default” for normal programming?

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

For a long time we’ve had heterogeneous devices on the embedded front (even before GPGPUs, DSPs, image co processors). We’ve never achieved a good, uniform way of programming these menageries with high level abstractions. The hardware can do more to support a uniform programming environment, and we’ve seen aggressive movement in this direction from NVidia GPUs. Longer term, variations on reconfigurable architectures may provide an alternative to the divergence-and-specialization trend (e.g. a phone with ASICs for everything), which opens up its own serious questions of programming abstractions and infrastructure.

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

Multicore architectures with weak or non-coherent memory will make traditional shared memory parallel programming less desirable, but at the same time, purely message passing (e.g. MPI) is insufficient. Large non-volatile memories could change how we think about running process state vs. persisted state. Finally, if compiler technology for exotic parallel architectures crosses an (unknowable) threshold, then perhaps we can have more diversity in architecture (i.e. gambits like the Cell architecture would have been more successful).

4. **Methodologies**: How should we perform interdisciplinary research that spans applications, systems, and technologies? How should abstraction layers evolve?

Funding opportunities that combine application experts and tool researchers seem promising. (For example, the DARPA PPAML project fit this mold.) More programs like SI2 that focus on software deliverables could play an important role in encouraging researchers to go the last mile and release usable software. But this should not emphasize only
existing, widely used software, but should incentivize new projects to grow to a usable state.

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

The research-to-practice transition is always difficult, but with parallel programming it has proved especially challenging to -- not just do the basic scientific research -- but also have an impact on programmer behavior. In part, this is the usual problem of ideas in programming languages and models being slow to percolate – but it seems to be compounded by problems with deploying ideas about parallel programming models. Parallelism remains an unsafe proposition for much software, research prototypes are often unreliable, and parallel programming ideas coming from the research world don’t compose with each other or integrate easily.