Applications:
The trite answer for what applications we are interested in is that we'll be more interested in “big data” applications—however, that isn't all that interesting because it’s a catch-all term for a fairly wide range of applications with differing characteristics, united, at best, by working over large inputs. Some big data applications are no more than scaled up applications of the sort that application writers have understood how to parallelize and optimize for years—linear algebra-style computations—while other applications are embarrassingly parallel, while still others operate over complex, changing graphs. I think the more important point to make here is that we will be dealing with an incredibly wide variety of applications with very different behaviors, and that means that we probably will not be able to adopt a “one-size-fits-all” solution to the problems of scaling these applications up. The best we can hope for is solutions that are broad enough to handle a class of applications (e.g., a language for dealing with graph applications; a compiler that understands how to optimize tree applications, etc.)

Systems:
I think an interesting trend in hardware is that, in the pursuit of efficiency, hardware is becoming increasingly specialized. I mean this not in the sense of special-purpose accelerators (though the rise of such devices is an interesting problem in its own right), but in the sense that even hardware that is intended to be general is much more restricted than in the past. The obvious examples here are GPGPUs and vector units: they both aim to be relatively general and programmable while providing efficient parallelism, but they place fairly strong constraints on usage to achieve that efficiency. If you can exploit the hardware properly, it can give you tremendous amounts of parallelism with much more efficiency than, say, a large shared-memory system. But exploiting that hardware is challenging—writing programs that execute efficiently on vector units or GPUs is quite difficult. That leads to two potential design directions. First, we could investigate the hardware side of things: can we make the hardware somewhat more flexible, to make it easier to write programs that exploit it, without sacrificing efficiency? What’s the right tradeoff there? Second, we could investigate the software side of things: how can we make it easier to write applications that take advantage of this specialized hardware? I think this will boil down to language/compiler/runtime solutions that target classes of applications, as alluded to above.

Technologies:
I don't have much experience in this space. Though I would certainly be interested to understand what sorts of technologies are on the horizon (or are just coming to the fore, such as NVM, stacked memory, etc.). What's interesting to me is that they may not remove constraints, but instead add new ones. Will we have to change the way that we write programs to fully exploit these new technologies? I think it’s likely that any new technology we develop will make writing programs more challenging, not less.