Programming Environments

Final Priority Research Directions

Patrick McCormick (lead)
Barbara Chapman
John Mellor-Crummey
Michelle Strout
Alex Aiken
John Shalf
Mary Hall
Programming Environments

PRD 1.1 - **Programmability**: Need to simplify programming and enable performance portability (write once, and high performance everywhere). This implies much more sophisticated implementations: (1) Higher-level, orthogonal, and layered programming abstractions (descriptive and prescriptive), (2) new compiler technology, DSLs, autotuning hardware and software mapping mechanisms (ref PRD1.3), (3) hierarchical standardization of interfaces (both hardware and software) to support the hierarchical abstractions. *Why? In EH systems, the number of different technologies exceeds the ability of individuals to master them. Need to generate high performance code for diverse execution engines.*

PRD 1.2 - **Mapping**: Because of the diversity and complexity of EH machines, need to separate the writing of the program (ref PRD 1.1) from how it is mapped on to the machine. This will require performance introspection, both hardware and software mechanisms for online performance introspection and analysis to enable automation of dynamic optimization, and AI assistance across a range of architectures.

PRD 1.3 - **Data Centricity**: In EH systems, almost all operations are performed by accelerators. Data movement to and from accelerators is far more expensive than compute, and accelerators will generally have very limited memory that needs to be treated as a scarce resource. Hardware accelerator efficiency is undercut by data movement costs and suboptimal placement. EH programs should not hide the overall structure of how a program uses its data; need to make this more explicit for the underlying implementation to exploit. This is an issue that cuts across the entire software stack, from programming abstractions, mapping, compiler optimizations, down to runtime systems and memory protection/security mechanisms.

PRD 1.4 - **Correctness**: Formal methods and tools for synthesizing, testing, and proving the correctness of parallel programs. *Why: Because of code complexity for EH systems and extreme parallelism, manual verification will no longer be feasible given the combinatorial explosion of error possibilities. Establishing an appropriate definition of “correctness” in the context of diverse/heterogeneous hardware/algorithms (neuromorphic, quantum, etc…) is a more fundamental research challenge.*