# Dagstuhl-Seminar 07361
# Programming Models for Ubiquitous Parallelism

David Chi-Leung Wong, Albert Cohen, María J. Garzarán
Christian Lengauer, and Samuel P. Midkiff

Over the last three decades most of the increase in CPU performance has been achieved through higher clock speeds. These increases have stopped because of physical constraints such as heat dissipation, power consumption and current leakage. Simultaneously, the push for more complex out-of-order superscalar cores has also stopped, due to diminishing returns and poor performance per Watt efficiency. To increase performance, future and emerging processors, both general purpose and embedded, use architectures with multiple cores on a single chip. Current general-purpose chips range from conservative SMP models, like the Power4 and Power5 and the Itanium2 Montecito, to more innovative low-power designs like the Intel Core Duo (Yonah) and the heavily threaded 8-core UltraSparc T1 (Niagara). Distributed-memory multi-cores are emerging as well, like the Cell processor from IBM, and a variety of multi-VLIW embedded designs like the TI C64x or ST2x0 families. As parallelism comes to the masses, software development is running into a crisis: no longer can software developers count on binary-compatible performance improvements over time, and no longer can they think about a relatively simple sequential environment. Instead, they need to exploit parallelism for future performance gains. Can this thirst for parallel programming be quenched with current technologies? What are the research challenges ahead to improve productivity, scalability, efficiency and reliability of general-purpose and embedded parallel programming?

It is essential that the appropriate tools be provided to give developers the ability to generate efficient parallel applications. These tools should also help "heroic programmers" to intervene in the parallelization and optimization process, as target architectures depart more and more from the von Neumann model (whose portability will be regretted). The generation of these tools will require the cooperative support of the five pillars of applications development and deployment: (1) programming languages, (2) compilers and their integrated development environments (IDEs), (3) libraries, (4) middleware and operating systems, and (5) hardware architectures.

The goal of the seminar is to present a broad view of efforts across these five pillars in the interest of developing collaborative solutions that leverage the strengths of different elements in the stack of software and hardware that is a computer system. Topics of interest include:

**Programming Languages.** What are suitable programming models for targeting parallel machines? How can languages and IDEs support the expression of parallelism while helping in the development of correct programs? How can the convenience of shared memory programming be combined with the scalability, encapsulation and control of distributed memory programs? What are the most attractive models beyond the sequentially-consistent shared memory model, in terms of abstraction, scalabilty, and compiler friendliness? Is it reasonable to base a parallel programming model on speculative transactions only? What are the design challenges for embedded systems, and for real-time or safety-critical systems?

**Compilers and IDEs.** What are the limits of compilers in compiling these programs effectively and providing both good performance and high productivity, and what can be done to approach these limits? How can compilers rely on library, middleware, operating system and hardware support to generate code and detect errors? Are the optimization problems so complex that programmers should be given more access to compiler internals? Can this be compatible with portability of performance and high productivity? Are run-time and just-in-time compilation techniques part of the answer? Are the motivations for dynamic program generation going to increase together with the amount of parallelism on a chip?

**Libraries.** How can parallel libraries be automatically generated for different parallel machines? How can libraries provide high level abstractions to support compilers and to allow programmers to exploit parallel machines while targeting a simple or familiar set of operations similar to those found in sequential programs? What are the APIs needed for common, non-scientific and scientific applications? How can libraries be "self-aware" in monitoring their use by applications, and use this information for dynamic tuning and error detection?

**Middleware and Operating Systems.** How can middleware and operating systems provide support for the detection and elimination of common errors like deadlocks, races and performance bottlenecks? How can middleware and operating systems provide abstractions to support easy-to-use programming models? What are the scalability challenges for operating systems running on massively parallel chip multi-processors? What kind of run-time systems can harness the heterogeneous, massively parallel resources of future system-on-chip designs?

**Hardware Architectures.** How can hardware monitoring, transactional memory and speculation support be used to enhance the usability of memory models and debugging of parallel programs? What additional hardware features are desirable from the viewpoint of the software stack and programming model? How can "surplus" processor cores be used for debugging and reliability support? What level of reconfigurability (FPGA to network-on-chip) will be most sensible for general-purpose computing? For numerical codes? For embedded systems?