Are compiler code generation techniques going to transition along with the hardware transition from multi-core to many-core and hybrid systems and at what speed?

- **Single Core**
  - Classic performance optimizations

- **Multi-Core**
  - Auto-parallelization
  - Alternate code paths
  - Directive extensions

- **Many-Core**
  - Fine-grained parallelization
  - Speculative parallelization

- **Hybrid-Core**
  - Cooperating processor-specific code generators
  - Assembly of general and special purpose elements
What information do you need from a Compiler Intermediate Format to efficiently utilize multi-core, many-core and hybrid systems that is not available from traditional languages like C, C++, or F90? Are you looking at directive-based or library-based approaches or is there another approach that you like?

- Data that helps compilers improve multi-core utilization include: aliasing assertions, levels for pointers, variable scoping, expected runtime environment (processors, architectures, caches, cores, binding of threads)
- Multiple approaches will evolve over time
  - Directive-based: standard and extended OpenMP (e.g., auto-scoping of variables), Transactional memory
  - Library-based: various commercial and open-source options (Boost, Sun MCFX, Intel TBB)
  - Other: just-in-time (binary) parallelization, profile feedback, compiler commentary review
Is embedded global memory addressing (like Co-Array Fortran) to be widely available and supported even on distributed memory systems?

- Steps for adoption of global memory addressing:
  - Standard syntax defined and supported on a majority of vendor compilers
  - Tuned interactions with network run-time layer
  - Wide-spread use by ISVs in their library offerings
  - Knowledgeable programmers

- Alternatives with no compiler impact and less topology sensitivity:
  - MPI library
  - Job schedulers (Sun Grid Engine, LSF, etc.)
  - Map-Reduce (Hadoop)
  - Others
What kind of hybrid systems or processor extensions are going to be supported by your compiler's code generation suite?

- Sun's compilers support latest
  - UltraSPARC processors with 64 threads (8 cores x 8 h/w threads)
  - Intel and AMD processors with various SSE instructions
- Hybrid transactional memory* support in Sun's tool suite is planned for release in 2009
  - Hardware assistance (new instructions and microarchitectural structures) is being implemented in a future Sun processor
- Co-processors
  - GPGPU support is being considered

* Transactional memory is the ability to perform a set of instructions atomically without expensive atomic instructions
What new run-time libraries will be available to utilize multi-core, many-core, and hybrid systems and will they work seamlessly through dynamic linking?

- First, basic OS libraries need to be MT-safe & tuned for many-core systems
  - Libc, libm, etc.
  - Memory allocation (mt-malloc)
  - Hybrid transactional memory routines
- Next, domain-specific libraries can extend system libraries
  - Solvers (BLAS, LAPACK, ...)
  - C++ STL, Boost, etc.
  - Berkeley “Dwarfs”
- Possible resource contention when using multiple libraries