**Problem**
- Traditional compilers treat MPI (Message Passing Interface) calls as "black boxes"
- Opportunities for optimizing the calls and surrounding code are lost

**Solution**
- Boost compiler's knowledge of MPI
- Implement compiler transformations, apply to MPI calls in parallel application codes
- Result: optimized transformed code

Our AToMS approach provides the transformation in particles:

- **Variable Cloning**
  - Similar to register renaming
  - Data dependencies that impair code motion can be removed by inserting clones of dependent variables

- **Native Data Structure Transformation**
  - Commonly, MPI data structures mirror native data structures; processes send entire instances of structure to other instances via MPI
  - In some cases, users omit unused fields in MPI data structure
  - This creates non-contiguous data, forcing analysis for buffer placement
  - Optimize by arranging layout of native data structure at compile time
  - Put non-transmitted first or after transmitted fields
  - Adjust user's logical layout of MPI data structure

- **Communication Library Specific Transformations**
  - Use specialized communication libraries in place of MPI
  - Better use of network capabilities

- **MPI Collective Call Decomposition**
  - Software-based collective calls are implemented as sequence of point-to-point operations
  - Compiler can optimize this sequence inlined into program by overlapping individual transfers with computation

- **Code Motion for Overlap Window Expansion**
  - Move non-blocking, data transfer initiation calls to beginning of code
  - Move transfer termination calls towards end of code

---

**Poster:** Marcos Portnoi