MATLAB D
Compiling MATLAB to Parallel Fortran

Mary Fletcher, Cheryl McCosh, Guohua Jin, Ken Kennedy
{maryf, chom, jin, kennedy}@cs.rice.edu
Rice University

Parallel MATLAB

MATLAB is a popular language for scientific applications because it is **high-level** and **domain-specific**. However, MATLAB is not a high-performance language. Scientists typically develop algorithms in MATLAB, and then need to hand-translate to a lower-level language to obtain the necessary performance.

This problem is compounded by the fact that many scientific applications involve computation on large data sets that can benefit from parallel architectures. Additionally, there is limited number of expert programmers capable of optimizing to the increasingly complicated architectures.

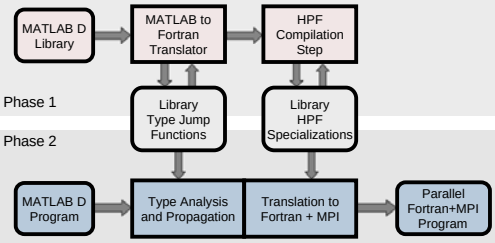
To allow scientists to easily write high-performance, parallel applications for their domains, **we are developing a compiler to automatically generate parallel Fortran code from MATLAB scripts**.

Telescoping Languages

Strategy **precompiles libraries**, generating variants specialized for different calling contexts, creating a **library-aware optimizer**. During user script compilation, library calls are quickly replaced by the best variant.

Telescoping Languages Strategy: Two Phase Compilation

Library Precompilation Phase
- Input: MATLAB D libraries (MATLAB plus distribution types)
- Perform type/distribution analysis and translate into Fortran
- Create optimized variants of routines specialized for input type and context
- Invoke High Performance Fortran compiler to generate communication in library routines
  - Automatically specializes new domain-specific libraries for different contexts and distributions
  - Easy to add new libraries



User Script Compilation
- Input: User MATLAB D script
- Perform type/distribution analysis
- Determine where communication is needed
  - User not responsible for communication
- Translate to Fortran and insert calls to data movement routines
  - Do not need to invoke HPF compiler
- Replace general routine calls with calls to specialized variants
  - Obtains performance comparable to optimizing library with user script

Benefits
- Fast compilation of user scripts
- Performance from domain knowledge
- Automatically creates new library-aware optimizer for new libraries

Base MATLAB Library
Routines need to:
- Be flexible for automatic specialization to different distributions and contexts
- Cover functionality of MATLAB

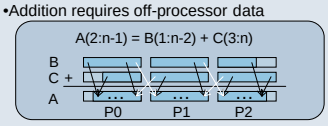
Communication generation
- Minimal requirement on writer for specifying parallelism
- Owner-computes rule
- Relying on HPF technology to generate communication

Type/distribution analysis
- Best Distribution
  - Minimize redistribution
  - Obtain fastest computation
- Data Movement Necessary

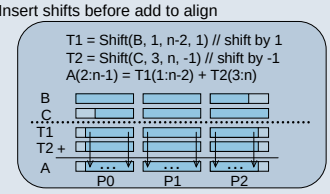
Related Strategies
- Specify parallelism by offering either:
  - MPI/PVM calls (too low-level)
  - Fixed distributions (miss optimization opportunities)
- Compile the large libraries with user script (slow compile times)

Example: User Script Compilation

Suppose **distribution analysis** yields Block distribution for A,B,C with offset 0 from template
- Addition requires off-processor data



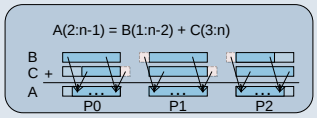
Insert shifts before add to align



Shifts are very expensive due to local copying
- Most data in T1 and T2 was copied locally

Instead, use **shadow regions** store a copy of off-processor data

Example requires a one-element shadow region for B and for C

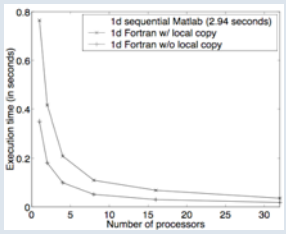


By using shadow regions in example,
- No local copying performed
- No temporaries needed to hold shifted array

Experimental Results for Example

Graph shows execution time of Fortran with and without the use of shadow regions
- Eliminating local copying improved performance up to factor of 2.7

Fortran versions on one processor are 3.8 and 8.4 times faster than MATLAB



Conclusion

The MATLAB D compiler generates high-performance, parallel Fortran code from MATLAB scripts with distribution information.

The telescoping languages framework automatically specializes libraries, quickly compiles user scripts, and obtains high performance from domain knowledge.

Implementation Status and Future Work

We are extending our **existing telescoping languages MATLAB compiler** to build the MATLAB D compiler. We have implemented a simple distribution analysis algorithm, which assumes moving the data at run-time to a new distribution is unnecessary. We are currently implementing algorithms to determine the best offset for each array from its template, and also the size of the overlap areas needed, if offsets alone cannot correctly align an array for all operations.

Once the offset and overlap analyses are implemented, we plan to focus on determining when an array should be redistributed at run-time and efficient methods for redistribution.

References

Chauhan, et al, "Automatic type-driven library generation for telescoping languages," SC '03.

Kennedy, et al, "Telescoping Languages: A system for automatic generation of domain languages," Proceedings of IEEE, 2005.

This material is based on work supported by the National Science Foundation under Grant No. CCF-0444465.