Generic Programming in POOMA and PETE

James A. Crottinger, Julian Cummings, Scott Haney, William Humphrey, Steve Karmesin, John Reynders, Stephen Smith, and Timothy J. Williams
Los Alamos National Laboratory; Los Alamos, NM 87545

Abstract. POOMA is a C++ framework for developing portable scientific applications for serial and parallel computers using high-level physical abstractions. PETE is an general-purpose expression-template library employed by POOMA to implement expression evaluation. This paper discusses generic programming techniques that are used to achieve flexibility and high performance in both POOMA and PETE. POOMA's array class factors the data representation and look-up into a generic engine concept. PETE's expression templates are used to build and operate efficiently on expressions. PETE is implemented using generic techniques that allow it to adapt to a variety of client-class interfaces, and to provide a powerful and flexible compile-time expression-tree-traversal mechanism.

1 Introduction

POOMA (Parallel Object-Oriented Methods and Applications) is an object-oriented framework for developing scientific computing applications on platforms ranging from laptops to parallel supercomputers [12]. POOMA includes C++ template classes representing high-level mathematical and physical abstractions such as arrays, particles, and fields. POOMA objects can be used in data-parallel expressions, with the parallelism encapsulated in the underlying framework. Expression creation and manipulation facilities are provided by PETE, the Portable Expression Template Engine.

This paper discusses generic programming techniques used to achieve flexibility and high performance in POOMA II and in PETE. POOMA II, currently under development, is a redesign of POOMA intended to further increase expressiveness and performance. POOMA arrays (the "II" will henceforth be understood) delegate data allocation and element access to a new engine class, allowing the array class to provide a uniform interface, including array expression

\[\text{This paper describes POOMA and PETE at the time of the Dagstuhl Seminar. Significant developments have occurred in the interim. The latest releases, along with examples and user-level tutorials, are available on the World Wide Web at http://www.acl.lanl.gov/pooma and http://www.acl.lanl.gov/pete. Certain important differences will be footnoted.}\]
capability, for a variety of data formats. Using PETE, POOMA separates the representation of an expression from its evaluation, allowing POOMA to provide multiple expression evaluation mechanisms. The simplest mechanism inline the entire evaluation in a manner similar to conventional expression-template array classes. Alternatively, an expression can be subdivided into expressions on subdomains of the arrays, and these sub-expressions can be evaluated independently by multiple threads.

PETE uses generic techniques to avoid assumptions about client-class interfaces and to provide a powerful and flexible expression tree traversal mechanism.

This paper is organized as follows. Section 2 discusses the engine abstraction. Section 3 gives a brief introduction to expression templates. Section 4 describes the use of PETE to add expression template capability to client classes, and the method by which PETE performs expression object manipulations, including evaluation. Section 5 concludes the paper with a discussion of POOMA’s expression-engine, an engine that allows expressions to be used as arrays.

2 Arrays and Engines

Many scientific computing applications require data types with multidimensional array semantics, but with a variety of underlying data structures. Examples range from regular arrays having Fortran or C storage order, to banded or sparse matrices, to array-like objects that compute their elements directly from their indices. One could model these data types using an inheritance hierarchy. An abstract base-class would define the interface, and descendent classes would override virtual functions to deal properly with their internal data structures. Unfortunately, the cost of virtual function calls in the inner loop of an array expression is prohibitive. Compile-time techniques are required to satisfy the performance requirements of most scientific applications.

POOMA’s Array class achieves the desired flexibility by factoring the data representation into a separate engine class. The Array provides the user interface and expression capability, while the engine provides data storage and mapping of indices to data elements.

All engines are specialization of an Engine template class:

```cpp
template <int Dim, class T, class EngineTag> class Engine { };
```

For example, a brick-engine, which stores a contiguous block of data that is interpreted as a multi-dimensional array with Fortran storage-order, is declared:

```cpp
class Brick {}

template <int Dim, class T> class Engine<Dim,T,Brick>;
```

The Brick class serves as a policy tag to choose a particular specialization of the Engine template. Requiring all engines to be specializations of a general Engine template allows for somewhat tighter type-checking, since functions that operate on engines in general can be given signatures of the form: