Exascale Computing Without Templates

Karl Rupp, Richard Mills, Barry Smith, Matthew Knepley, Jed Brown

Argonne National Laboratory

DOE COE Performance Portability Meeting, Denver
August 22-24, 2017
PETSc Developers Care About Recent Developments

After careful evaluation: Favor MPI 3.0 (and later) over threads
Find the best long-term solutions for our users
Consider best solutions for large-scale applications, not just toy-apps
Our Attempts in C++ Library Development

- Sieve: Several years of C++ mesh management attempts in PETSc
- ViennaGrid 2.x: Heavily templated C++ mesh management library
- ViennaCL: Dense and sparse linear algebra and solvers for multi- and many-core architectures
Providing Context

Our Attempts in C++ Library Development

Sieve: Several years of C++ mesh management attempts in PETSc
ViennaGrid 2.x: Heavily templated C++ mesh management library
ViennaCL: Dense and sparse linear algebra and solvers for multi- and many-core architectures

Aftermath

Sieve: Replaced by DMPlcx (written in C)
ViennaGrid: Version 3.0 provides C-ABI
ViennaCL: Rewrite in C likely

Sequential build times for the ViennaCL test suite
Disadvantages of C++ Templates

Static Dispatch

- Architecture-specific information only available at run time
- “Change code and recompile” not acceptable advice
Disadvantages of C++ Templates

Static Dispatch

- Architecture-specific information only available at run time
- “Change code and recompile” not acceptable advice

Dealing with Compilation Errors

- Type names pollute compiler output
- Replicated across interfaces
- CRTP may result in type length explosion
- Default arguments become visible

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>std::vector&lt;int&gt;</code></td>
<td>38</td>
</tr>
<tr>
<td><code>std::vector&lt;std::vector&lt;int&gt;&gt;</code></td>
<td>109</td>
</tr>
<tr>
<td><code>std::vector&lt;std::vector&lt;std::vector&lt;int&gt;&gt;&gt;</code></td>
<td>251</td>
</tr>
<tr>
<td><code>std::vector&lt;std::vector&lt;std::vector&lt;std::vector&lt;int&gt;&gt;&gt;&gt;</code></td>
<td>539</td>
</tr>
</tbody>
</table>

https://xkcd.com/303/
The Grand C++ Error Explosion Competition

Results of tgceec 2015

The deadline for the Grand C++ Error Explosion Competition has passed. We would like to thank all those who participated.

However, we received only a very small number of entries. Because of this we have decided to cancel the competition. Due to this lack of interest we suspect that the competition will not be run again next year.

https://tgceec.tumblr.com/
Disadvantages of C++ Templates

Scope Limitations

Template metaprogramming lacks state
Optimizations across multiple code lines difficult or impossible

Example
Consider vector updates in pipelined CG method:

\[
\begin{align*}
  x_i &\leftarrow x_i - 1 + \alpha p_i - 1 \\
  r_i &\leftarrow r_i - 1 - \alpha y_i \\
  p_i &\leftarrow r_i + \beta p_i - 1
\end{align*}
\]

Reuse of \( p_i - 1 \) and \( r_i - 1 \) easy with for-loops, but hard with expression templates
Disadvantages of C++ Templates

Scope Limitations

Template metaprogramming lacks state
Optimizations across multiple code lines difficult or impossible

Example

Consider vector updates in pipelined CG method:

\[
\begin{align*}
x_i & \leftarrow x_{i-1} + \alpha p_{i-1} \\
r_i & \leftarrow r_{i-1} - \alpha y_i \\
p_i & \leftarrow r_i + \beta p_{i-1}
\end{align*}
\]

Reuse of \( p_{i-1} \) and \( r_{i-1} \) easy with for-loops, but hard with expression templates
Disadvantages of C++ Templates

Complicates Debugging

- Stack traces get longer names and deeper
- Setting good breakpoints may become harder
Disadvantages of C++ Templates

Complicates Debugging

- Stack traces get longer names and deeper
- Setting good breakpoints may become harder

Lack of a Stable ABI

- Object files from different compilers generally incompatible
- Name mangling makes use outside C++ land almost impossible
Disadvantages of C++ Templates

Complicates Debugging
- Stack traces get longer names and deeper
- Setting good breakpoints may become harder

Lack of a Stable ABI
- Object files from different compilers generally incompatible
- Name mangling makes use outside C++ land almost impossible

High Entry Bar
- Number of potential contributors inversely proportional to code sophistication
- Domain scientists have limited resources for C++ templates
A Path Forward

Manage Complexity

Good interface design
Refactor code when needed
Hand-optimize small kernels only (cf. BLIS methodology)

Fig. 1. An illustration of the algorithm for computing high-performance matrix multiplication, as expressed within the BLIS framework [Van Zee and van de Geijn 2015].

Development Implications

Adopt professional software development practices
Develop, maintain, and evolve different datastructures and code paths
Use clear and easy-to-understand datastructures
Fallacy: “Writing” an application only once in its final form
A Path Forward

Manage Complexity

Good interface design
Refactor code when needed
Hand-optimize small kernels only (cf. BLIS methodology)

Development Implications

Adopt professional software development practices
Develop, maintain, and evolve different datastructures ... ... and code paths
Use clear and easy-to-understand datastructures
Fallacy: “Writing” an application only once in its final form

Fig. 1. An illustration of the algorithm for computing high-performance matrix multiplication, as expressed within the BLIS framework [Van Zee and van de Geijn 2015].

ACM Transactions on Mathematical Software, Vol. 0, No. 0, Article 0, Publication date: 2016.

[F. Van Zee, T. Smith, ACM TOMS 2017]
Spending Development Resources

- Reuse existing libraries — reinventing the wheel is not productive!
- Focus on domain- and application-specific aspects
- Obtain expertise and resources for continuous code evolution
A Path Forward

Spending Development Resources

- Reuse existing libraries — reinventing the wheel is not productive!
- Focus on domain- and application-specific aspects
- Obtain expertise and resources for continuous code evolution

Required Incentives

- Reward contributions to existing projects
- Pair research funding with software development funding
- Establish software development career tracks
Is Performance Portability Just a Software Productivity Aspect?
Summary

Long-Term Problems of Heavy C++ Templates Use

- Template metaprogramming is a leaky abstraction
- Excessive type names slow down all stages of Compile-Run-Debug-cycle
- Templates operate at compile time - architecture ultimately known at run time

A Path Forward

- Adopt professional software development practices
- Be prepared to develop different datastructures and code paths
- Write clear, readable code using simple datastructures
- Evolve and refactor datastructures, kernels, and interfaces over time
  *(cf. software productivity discussions)*