Analyses of How Code Organization Impacts Development-Time Process

Damian Rouson
Sandia National Laboratories

Joel Koplik, Xiaofeng Xu, Karla Morris
City University of New York

Robert Rosenberg
U.S. Naval Research Laboratory

Stavros Kassinos, Irene Moulitsas
University of Cyprus

Sponsors:
Office of Naval Research, National Science Foundation, European Commission
Motivation

Code writing, efficiency & translation
Limits of HPC software tools
Personnel
Hardware
MPI issues
Performance
I/O issues
Other

* Note: Multiple responses allowed.
Objectives

1. To analyze how development time scales with program size & how this depends on the choice of abstractions.
2. To develop strategies for reducing development time.
3. To demonstrate *scalable development* of multiphysics models.
Outline

1. Analyses of the impact of
   a. Coding efficiency on total solution time.
   b. Programming paradigm on debugging.
   c. Abstraction choice on interface content.
2. Multiphysics model demonstrations
3. Conclusions & Future Work
Outline

1. Analyses of the impact of
   a. Coding efficiency on total solution time.
   b. Programming paradigm on debugging.
   c. Abstraction choice on interface content.
2. Multiphysics model demonstrations
3. Conclusions & Future Work
Amdahl’s Law

Representative case study for a published run\textsuperscript{1,2}:

\[
\begin{aligned}
\text{Code Writing Time} & \quad \text{Debugging Time} \quad \text{Run Time} \\
\frac{1}{3} & \quad \frac{1}{3} & \quad \frac{1}{3}
\end{aligned}
\]

Run-time speedup: \( S_{\text{run}} \equiv \frac{\text{original run time}}{\text{optimized run time}} \)

Total speedup: \( S_{\text{tot}} = \frac{1}{\frac{2}{3} + \frac{1}{3} S_{\text{run}}} \Rightarrow \lim_{S_{\text{run}} \to \infty} S_{\text{tot}} = 1.5 \)

The speedup achievable by focusing solely on decreasing run time is very limited.

\textsuperscript{1}Rouson et al. (2008) *Physics of Fluids*.
\textsuperscript{2}Rouson et al. (2008) *ACM Transactions on Mathematical Software*. 
Case Study: Isotropic Turbulence

- 5% procedures occupy nearly 80% of run time.
- Structure 95% of procedures to reduce development time.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Inclusive Run-Time Share (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>100.0</td>
</tr>
<tr>
<td>operator(.x.)</td>
<td>79.5</td>
</tr>
<tr>
<td>RK3_Integrate()</td>
<td>47.8</td>
</tr>
<tr>
<td>Nonlinear_Fluid()</td>
<td>44.0</td>
</tr>
<tr>
<td>Statistics_</td>
<td>43.8</td>
</tr>
<tr>
<td>transform_to_fourier</td>
<td>38.7</td>
</tr>
<tr>
<td>transform_to_physical</td>
<td>23.6</td>
</tr>
<tr>
<td>Number of Processors</td>
<td>Total Solution Time</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>1</td>
<td>1.25</td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>3</td>
<td>1.75</td>
</tr>
</tbody>
</table>

Theoretical Limit

Intel Math Kernel Library (MKL)

SGI Math Library
Pareto Principle

When participants (lines) share resources (run time), there always exists a number \( k \in [50, 100] \) such that \((1-k)\%\) of the participants occupy \(k\%\) of the resources:

Limiting cases:

- \(k=50\%, \) equal distribution
- \(k \rightarrow 100\%, \) monopoly

**Rule of thumb:** 20% of the lines occupy 80% of the run time

Scalability requirements determine the percentage of the code that can be focused strictly on programmability:

\[
S_{\text{max}} = \lim_{{S_k \rightarrow \infty}} \frac{1}{0.2 + 0.8 / S_k} = 5
\]
Outline

1. Analyses of the impact of
   a. Coding efficiency on total solution time.
   b. Programming paradigm on debugging.
   c. Abstraction choice on interface content.

2. Multiphysics model demonstrations

3. Conclusions & Future Work
“Separate the physics from the data.”

Jaideep Ray

Sandia National Laboratories, ca. 2005
“Software abstractions should resemble blackboard abstractions.”

Kevin Long
Texas Tech. Univ., ca. 2007
Abstract Data Type Calculus

Blackboard abstraction

\[ T = T(x, y, z, t) \]

\[ T(x = 0, y, z, t) = T_0 \]

\[ T_t \equiv \frac{\partial T}{\partial t} \]

\[ T^{n+1} = T^n + \Delta t T^n_t \]

\[ \frac{\partial T}{\partial t} = \frac{1}{\alpha} \nabla^2 T \]

\[ \nabla^2 T \equiv T_{xx} + T_{yy} + T_{zz} \]

Software abstraction (Fortran 2003):

\[
\text{type(field)} :: T, dT\_dt, \text{laplacian}\_T \\
\text{call T\%boundary(x, 0, T0)} \\
T\%t() \\
T = T + dt\*T\%t() \\
\text{dT}\_dt = (1./alpha)\*\text{laplacian(T)} \\
\text{laplacian}\_T = T\%xx() + T\%yy() + T\%zz() 
\]
Abstract Data Type (ADT)

module field_class
    implicit none
    private
    type, public :: field
        private
        real, dimension(:,:,,:), allocatable :: nodalValues
    contains
        procedure :: boundary
        procedure :: plus
        generic, public :: operator(+) => plus
    end type
contains
    subroutine boundary(this,direction,location,value)
        class(field) :: this
        ...!
    end subroutine
    function plus(lhs,rhs) result(total)
        class(field), intent(in) :: lhs,rhs
        class(field), allocatable :: total
        ...!
    end function
“Procedural programming is like an N-body problem.”

Lester Dye, Stanford University, ca. 1994
“What are the metrics?”
Oyekunle Olukotun, ca. 1996
Stanford University
“Not much time is spent fixing bugs. Most of the time is spent finding bugs.”

Shalloway & Trott (2002) *Design Patterns Explained*
Debugging Structured Programs

program main
real :: T(100), alpha=1., dt=0.1, dx=0.01
T = T + dt*(1./alpha)*laplacian(T)

function laplacian(T)
real :: T(:), diff(size(T),3), laplacian(size(T))
laplacian(:)=diff(:,1)*T(:)+diff(:,2)*T(:)+diff(:,3)*T(:)

Legend
→ Write
→ Read
Fault Localization

“Computational” complexity theory: Derive a polynomial time estimate for fault localization in a chronological list of the unique program lines executed:

\[ \frac{\lambda}{2} - 1 \]
\[ \frac{(\lambda/2 - 1)}{2} \]

\[ \lambda \]

\[ \alpha < 0 \quad \text{(bug)} \]
\[ T(2) < 0 \quad \text{(symptom)} \]
Fault Rate

Faults/line (x1000)

Module size (lines)

$\Rightarrow r \approx 6/1000$

Scientific Code Faults

Observed faults in *commercially released code*:  
• 8 statically detectable faults/1000 lines of C code  
• 12 statically detectable faults/1000 lines of Fortran 77 code  
• more recent data finds 2-3 times as many faults in C++

\[
\Rightarrow r \approx 0.006 - 0.036 \\
t_{\text{search}} = \left( \# \text{ bugs} \right) \times \left( \text{lines searched per bug} \right) \left( \bar{t}_{\text{line review}} \right) \\
= (r\lambda) \left[ (\lambda/2 - 1)/2 \right] \left( \bar{t}_{\text{line review}} \right)
\]

Bisection Search Time

Localization error:
\[ \lambda_n = \frac{\lambda}{2^n} \]

Convergence criterion:
\[ \frac{\lambda}{2^q} = 1 \]
\[ q = \log_2 \lambda \]

Search time metric:
\[ \lambda_{\text{searched}} = r\lambda \log_2 \lambda \]
\[ \lambda_{\text{searched}} = r \lambda_m \log_2 \lambda_m \]

\[ \lambda_m \ll \lambda \]
Procedural line density:

\[ \rho = \frac{\lambda_m}{p} = \frac{\text{lines per module}}{\text{procedures per module}} \]

For ADT calculus, \( \rho \approx \text{const} \)

\[ p \approx \text{const} \]

\[ \Rightarrow \lambda_{\text{searched}} = (r \rho p) \log_2 \rho p \]

\[ \Rightarrow \lambda_{\text{searched}} \approx \text{const} \]
Outline

1. Analyses of the impact of
   a. Coding efficiency on total solution time.
   b. Programming paradigm on debugging.
   c. Abstraction choice on interface content.
2. Multiphysics model demonstrations
3. Conclusions & Future Work
Abstract class interface (Unified Modeling Language):

<table>
<thead>
<tr>
<th>integrable_model</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ operator(+) (integrable_model, integrable_model) : integrable_model</td>
</tr>
<tr>
<td>+ operator(*) (real, integrable_model) : integrable_model</td>
</tr>
<tr>
<td>+ t(integrable_model) : integrable_model</td>
</tr>
</tbody>
</table>

A single interface describes all of the public information for all classes that extend this class.
Information Entropy

Shannon (1948) “A mathematical theory of communication,” *Bell System Tech. J.*

The class interfaces embody inter-developer communications. Consider the set of all \( N \) possible messages that can be transmitted between two developers:

“If the number of messages in the set is finite, then this number or any monotonic function of this number can be regarded as a measure of the information produced when one message is chosen from the set, all choices being equally likely.”

Shannon chose the logarithm because it satisfies several constraints that match our intuitive understanding of information:

\[
H = - \sum_{i=1}^{N} p_i \log_2 p_i = - \sum_{i=1}^{N} \frac{1}{N} \log_2 \frac{1}{N} = \log_2 N
\]
Minimum Information Growth

subroutine integrate(integrand)
  class(integrable_model) :: integrand
  integrand = integrand + dt*integrand%t()
end subroutine

If only one class extends integrable_model, the executable line only has one possible interpretation, so $H=0$. Each subsequent subclass increases the information content by

$$\Delta H = \log_2 (N + 1) - \log_2 N$$

which is the minimum information growth.
Outline

1. Analyses

2. Multiphysics model demonstrations
   a. Particle transport in magnetohydrodynamics.
   b. Quantum turbulence in superfluid $^4$He.
   c. Atmospheric boundary layer.
   d. Lattice-Boltzmann bio-fluid dynamics.

3. Conclusions & Future Work
Lattice Boltzmann bio-fluid dynamics:
Particle-Laden MHD

Strong magnetic fields damp velocity variations in the field direction, leading to 2D/3C state:

\[
\frac{\partial}{\partial t} \tilde{u}(\tilde{x}, t) = \ldots + \frac{1}{\eta} \nabla^2 \left( B_{A}^{ext} \cdot \nabla \right)^2 \tilde{u}(\tilde{x}, t) + \ldots
\]

Cross-stream dispersion segregates inertial particles:

\[
dr_i / dt = v_i \\
v_i / \text{St} = \left[ u_i(\vec{r}, t) - v_i \right] / \text{St}
\]

\[
\text{St} = \tau_p / \tau_f
\]
Quantum Turbulence

Below 2.17 K, liquid $^4$He flows as a two-fluid mixture with mutual friction between the two components:

1. Normal viscous fluid

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \vec{u} + \vec{f}$$

$$\nabla \cdot \vec{u} = 0$$

2. Inviscid superfluid with quantized vortex circulation

$$\kappa \equiv \hbar / m_{^4\text{He}}$$

$$\mathbf{v}_i = \frac{\kappa}{4\pi} \int (\mathbf{S}_0 - \mathbf{r}) \otimes d\mathbf{S}_0 / \|\mathbf{S} - \mathbf{r}\|^3$$

$$\frac{d\mathbf{S}}{dt} = \mathbf{v}_i + \alpha \mathbf{S}' \otimes (\mathbf{v}_n - \mathbf{v}_i) - \alpha' \mathbf{S}' \otimes [\mathbf{S}' \otimes (\mathbf{v}_n - \mathbf{v}_i)]$$
Quantum Turbulence

Quantum vortices driven by forced, isotropic normal-fluid turbulence at 2.1 K:
Vortex Locking

Quantum vortex alignment with classical vortices in frozen normal-fluid turbulence:

Conclusions

- Applying Amdahl’s law to the total solution time suggests that optimizing runtime only severely limits speedup.
- The Pareto Principle determines the percentage of the code that can be focused strictly on programmability rather than runtime efficiency.
- ADT calculus renders bug search times very nearly scale-invariant and reduces interface information content.
Future Directions

• Demonstrate runtime scalability.
• Add empirical support for reductions in
  1. Fault localization time.
  2. Information entropy*.

“First they ignore you. Then they ridicule you. Then they fight you. Then you win.”

Mahatma Ghandi
Traditional Design Metrics

Structured programming:
• Source Lines of Code (SLOC)
• Cyclomatic complexity

Object-Oriented Programming:
• Afferent couplings (Ca): # packages that depend on a given one.
• Efferent couplings (Ce): # packages a given package depends on.
• Instability: \[ I \equiv \frac{Ce}{Ce + Ca} \]