Can Scalable Development Lead to Scalable Execution?

Damian Rouson
Scalable Computing R&D
Sandia National Laboratories

Sponsors:
Office of Naval Research, City University of New York
Outline

• Motivation, Objectives & Guideposts
• Conventional development
• Scalable development
• Applications
• Toward scalable execution
• Conclusions & Acknowledgments
Motivation

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

What Are the Biggest Bottlenecks Today in Creating Custom Parallel Applications?*

* Note: Multiple responses allowed.
Objectives

1. To develop a design methodology that scales up to large numbers of programming units, e.g. procedures, classes, data structures, etc.
2. To demonstrate that this methodology can produce new science.
3. To demonstrate that this approach also scales up to large numbers of execution units, e.g. threads, processes, cores, etc.
“What are the metrics?”
Oyekunle Olukotun, Stanford EE/CS, c. 1996

“Procedural programming is like an N-body problem.”
Lester Dye, Stanford Petroleum Eng., c. 1995

“Separate the data from the physics.”
Jaideep Ray, Sandia, c. 2004

“First they ignore you. Then they laugh at you. Then they fight you. Then you win.”
Mahatma Ghandi, c. ????
Outline

• Motivation, Objectives & Guideposts
• Conventional development
  – Amdahl’s Law
  – Pareto Principle
  – Complexity
• Scalable development
• Applications
• Toward scalable execution
• Conclusions & Acknowledgments
Conventional Development

Total solution time

Mathematical Modeling → Code writing → Debugging → Production Run

Barrier

Sandia National Laboratories
Amdahl’s Law

Representative case study for a published run*:

\[ S_{tot} = \lim_{S_{run} \to \infty} \frac{1}{\frac{\frac{1}{3}}{2} + \frac{\frac{1}{3}}{3} S_{run}} = 1.5 \]

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

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
- $k \to 100\%$, 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:

$$S_{\text{max}} = \lim_{S_k \to \infty} \frac{1}{0.2 + 0.8 / S_k} = 5$$
### Runtime Profile

<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>
</tbody>
</table>

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

Theoretical Limit

Intel Math Kernel Library (MKL)

SGI Math Library

Number of Threads

Total Solution Time Speedup

1.75
1.5
1.25
1.0

1 2 3 4 5 6 7 8

Number of Processors
Conventional Development

Model Problem: Unsteady 1D Diffusion

\[ \frac{\partial \phi}{\partial t} = D \nabla^2 \phi \]

Semi-discrete equations:

\[ \frac{d}{dt} \phi_i = D \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \]

Fully discrete equations:

\[ \phi_i^{n+1} = \phi_i^n + \Delta t \cdot D \frac{\phi_{i+1}^{n} - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2} \]

Solution algorithm:

\[ \bar{\phi} \leftarrow \bar{\phi} + \frac{\Delta t \cdot D}{\Delta x^2} [A] \bar{\phi} \]
Conventional Program Debugging

“Not much time is spent fixing bugs. Most of the time is spent finding bugs.”
-- Shalloway & Truitt, Design Patterns Explained
-- Oliveira & Stewart, Writing Scientific Software

PROGRAM main
REAL :: phi(100), D=1., dt=0.1, dx=0.01
phi = phi + (D*dt/dx**2)*laplacian(phi)

FUNCTION laplacian(phi)
REAL :: phi(:), A(SIZE(phi),3), laplacian(SIZE(phi))
laplacian(:)=A(:,1)*phi(:)+A(:,2)*phi(:)+A(:,3)*phi(:)

phi(1), phi(2), ..., phi(100) Data Set

Legend
→ Write
→ Read
Bug Search Complexity

Consider a list of the unique program lines with all lines that execute before the symptom preceding the symptomatic line:

\[
\ell / 2 - 1 \quad \frac{(\ell / 2 - 1)/2}{D < 0 \quad \text{(bug)}}
\]

\[
\ell \quad \phi(2) < 0 \quad \text{(symptom)}
\]
Code Fault Rates


\[ r \approx \frac{6}{1000} \]
Scientific Code Fault Rates


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

\[ r \approx 0.006 - 0.036 \]

\[
t_{search} = (# bugs) \times (\text{lines searched per bug}) \left( t_{line review} \right)
\]

\[
= (r \ell) \left[ \left( \frac{\ell}{2} - 1 \right) / 2 \right] t_{line review}
\]
Outline

• Motivation, Objectives & Guideposts
• Conventional development
• Scalable development
  – Complexity
  – Information theory
• Applications
• Toward scalable execution
• Conclusions & Acknowledgments
Object-Oriented Programming

\[ t_{\text{search}} = (r \ell_m) \left( \frac{\ell_m}{2 - 1} / 2 \right) t_{\text{line review}} \]

\[ \ell_m \ll \ell \]
Scientific OOP

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

\[ t_{\text{search}} = (r \rho p) \left( \frac{\rho p}{2} - 1 \right) / 2 \frac{\bar{t}}{\text{line review}} \]
Decomposing the problem into a set of classes that admit an abstract data type calculus yields

\[
\rho \approx \text{const.}, \quad p \approx \text{const.}
\]
Information Theory

• Interface information content sets the minimum amount of communication between developers.

• Let \( p_i \) = frequency of occurrence of the \( i^{th} \) keyword in a set of statements. Shannon entropy is

\[
S = - \sum_i p_i \log p_i \geq 0
\]

• Repeated implementation of same procedural interfaces generates high \( p_i \) values \( \rightarrow \) low \( S \).

Kolmogorov Complexity

• For a program $p$, the Kolmogorov complexity $K(p)$ is the shortest description in some description language

• Properties:
  – Provably not computable.
  – Bounded from above by any actual description of $p$.
  – Lowest upper bound at any given time: compressed program length + decompression program length

• Using this measure, we have detected slightly greater complexity in C++ than Fortran 2003
## Applications

### Currently Running

<table>
<thead>
<tr>
<th>Solid particle dispersion in electrically conducting fluids*:</th>
<th>Quantum vortex interactions with classical fluids**:</th>
<th>Aerosol dispersion in the atmospheric boundary layer***:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Integrator</td>
<td>Time Integrator</td>
<td>Time Integrator</td>
</tr>
<tr>
<td>Mixture</td>
<td>Mixture</td>
<td></td>
</tr>
<tr>
<td>Cloud</td>
<td>Fluid</td>
<td>Cloud</td>
</tr>
<tr>
<td>Magnetofluid</td>
<td>Field</td>
<td>Scalar</td>
</tr>
<tr>
<td>Fluid</td>
<td>Field</td>
<td>Fluid</td>
</tr>
<tr>
<td>Field</td>
<td>Grid</td>
<td></td>
</tr>
<tr>
<td>Grid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Vertically adjacent layers communicate through interfaces.)


*** Rouson & Handler (2007) in *Environmental Sciences & Environmental Computing, Vol. III.*
Large Eddy Simulation of the ABL

Physical Processes

- Shear
- Buoyancy
- Coriolis effects
- Geostrophic wind forcing
- Thermal Fluctuations
- Passive Scalar

Code Details

- Fully spectral LES: Fourier in horizontal, Chebyshev in vertical.
- Uniform grid in horizontally, cosine-stretched grid vertically.
- Compressibility is neglected (different from COAMPS).
**Governing Equations**

**Momentum:**
\[
\frac{\partial \mathbf{u}}{\partial t} = \mathbf{N} + \mathbf{C} - \nabla \Pi + \nabla \cdot \tau_{sgs} + \frac{\theta'}{\theta_o} g \hat{e}_3 + \frac{dP}{dx} \hat{e}_1
\]

- **Advection**
- **Coriolis**
- **Pressure**
- **Subgrid Physics**
- **Buoyancy**
- **Geostrophic pressure gradient**

**Mass**
\[
\nabla \cdot \mathbf{u} = 0
\]

**Heat**
\[
\frac{\partial \theta'}{\partial t} + \mathbf{u} \cdot \nabla \theta' = \nabla \cdot \left( \frac{\nu_T}{Pr_T} \nabla \theta' \right)
\]
\[
\Pi \equiv \frac{c_p}{\gamma} \bar{\theta} \pi' + \bar{u} \cdot \bar{u}/2
\]

**Exner Function:**
\[
\pi \equiv \left( \frac{p}{p_0} \right)^{\gamma-1}
\]

**Virtual Temperature:**
\[
\theta \equiv \frac{T}{\pi}
\]

**Smagorinsky Sub-Grid Scale Turbulence Model:**
\[
\tau_{sgs} = 2 \nu_T \cdot S
\]
\[
\ell \sim \Delta \quad \text{(grid scale)}
\]
\[
\nu_T = C_s \ell^2 \sqrt{2 \cdot |S|^2}
\]
Simulation Parameters

• VERY SIMPLE PHYSICS
  • PRESSURE GRADIENT IN THE X-DIRECTION
  • CONSTANT TURBULENT VISCOSITY
  • ROTATING EARTH

\[ L_x = 12.57 \text{ km} \quad L_y = 2.0 \text{ km} \quad L_z = 4.71 \text{ km} \]

\[ G = 2.88 \text{ m/s (geostrophic wind)} \]
\[ \nu_T = 0.72 \text{ m}^2/\text{s (Agrees reasonably well with Sullivan et al BL Met. 1994)} \]
\[ \Omega_E = 7.2722 \times 10^{-5} \text{ rad/s} \]

THESE PARAMETERS GIVE

\[ \Delta_{Ekman} = \left( \frac{\nu_T}{\Omega_E} \right)^{1/2} = 0.1 \text{ km} \]

\[ Re = \frac{(G \times L_y)}{\nu_T} = 8000 \]
$U^*/G = 0.067$  In good agreement with Coleman et al (JFM, 1990)
Wind Velocity (W) in an x-y plane

Thin Ekman Layer with turbulent “eruptions”
W in x-z plane at 43 meters

Note highly elongated low speed regions and “gusts”
Vertical vorticity 640 meters in x-z plane

Note “coherent 2D vortices” --- Air-Spikes !?
Outline

• Motivation, Objectives & Guideposts
• Conventional development
• Scalable development
• Applications
• Toward scalable execution
  – A strategy
  – Turbulence at the petascale
• Conclusions & Acknowledgments
Toward Scalable Execution

class(Scalar) :: Smoke
Smoke = Smoke + dt*d_dt(Smoke)

Strategy:

• Decompose problem into elementary operations.
• Instantiate distributed objects, e.g. via Trilinos.
• Parallelize operators across distributed objects.

Potential pitfalls:

• Cache utilization.
• Combined instructions.
Turbulence at the Petascale

- R. D. Moser* estimates 1500 Petaflop-hours required for DNS at \( Re_\tau = 5000 \), which will achieve asymptotic behavior in the log layer.
- The bottom plane of many ABL simulations lies in the log layer & employs a boundary condition valid at asymptotically high Reynolds number:

\[
\begin{align*}
  u^+ &= \left( \frac{1}{k} + \frac{\beta}{Re_\tau} \right) \ln y^+ + \frac{\alpha y^+}{Re_\tau} + B \\
  \lim_{Re_\tau \to \infty} u^+ &= \frac{1}{k} \ln y^+ + B
\end{align*}
\]

Conclusions

• Applying Amdahl’s law to the total solution time suggests that optimizing run time only severely limits speedup.
• The Pareto Principle determines the percentage of code that can be focused on programmability rather than efficiency.
• The global data sharing in conventional development leads to a quadratic search times.
• Enabling an abstract data type calculus
  – Renders bug search times roughly scale-invariant and
  – Limits interface content (developer communications)
• We have demonstrated scalable development on several applications and proposed a path toward scalable execution.
Acknowledgements

• NRL
  – Dr. Robert Handler
  – Dr. Robert Rosenberg

• CUNY
  – Prof. Joel Koplik
  – Ms. Karla Morris & Dr. Xiaofeng Xu

• University of Cyprus
  – Prof. Stavros Kassinos
  – Dr. Irene Moulitsas, Dr. Evaggelos Akylas, & Dr. Hari Radhakrishnan

• University of Belgium
  – Prof. Bernard Knaepen
  – Dr. Ioannis Sarris