This is not Your Parents’ Fortran: A Scalable, Parallel, Functional OO PDE Solver

Damian Rouson

STANFORD UNIVERSITY
CENTER FOR COMPUTATIONAL EARTH AND ENVIRONMENTAL SCIENCES

CEED
In 1 academic quarter, I teach beginning graduate students how to write a

- Parallel
- Functional
- Object-Oriented

PDE solver using high-level mathematical abstractions,
Scaling beyond 16,000 cores with nearly 90% parallel efficiency,
Using either of 2 commercially released compilers or one pre-release open-source compiler,
With no reliance on libraries external to the language (e.g., no OpenMP or MPI in the source)
With zero chance of common beginner mistakes (e.g., no memory leaks or dangling pointers).
Fortran’s Image

http://www.computersciencelab.com/ComputerHistory/HistoryPt4.htm

http://www.computersciencelab.com/ComputerHistory/HistoryPt4.htm

http://longstreet.typepad.com/thesciencebookstore/computer_techhistory/

http://www.clemson.edu/caah/history/facultypages/PamMack/lec122sts/computers.html
Question 31: Which programming models and languages do you use for code development? Please select one or more from the following list.

Response rate: 78%

### Fortran’s Future

8. Please indicate your requirements for comprehensive formal training in the following programming languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>Not important</th>
<th>Somewhat important</th>
<th>Very Important</th>
<th>Rating Average</th>
<th>Response Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN77</td>
<td>75.9% (60)</td>
<td>16.5% (13)</td>
<td>7.6% (6)</td>
<td>0.32</td>
<td>79</td>
</tr>
<tr>
<td>Fortran 95</td>
<td>31.8% (27)</td>
<td>41.2% (35)</td>
<td>27.1% (23)</td>
<td>0.95</td>
<td>85</td>
</tr>
<tr>
<td>Fortran 2003</td>
<td>34.9% (29)</td>
<td>36.1% (30)</td>
<td>28.9% (24)</td>
<td>0.94</td>
<td>83</td>
</tr>
<tr>
<td>C</td>
<td>37.5% (33)</td>
<td>35.2% (31)</td>
<td>27.3% (24)</td>
<td>0.90</td>
<td>80</td>
</tr>
<tr>
<td>C++</td>
<td>38.9% (35)</td>
<td>33.3% (30)</td>
<td>27.8% (25)</td>
<td>0.89</td>
<td>90</td>
</tr>
<tr>
<td>Java</td>
<td>78.1% (57)</td>
<td>17.6% (13)</td>
<td>4.1% (3)</td>
<td>0.26</td>
<td>73</td>
</tr>
<tr>
<td>Scripting Languages (Python, PERL, Ruby etc.)</td>
<td>39.1% (34)</td>
<td>41.4% (36)</td>
<td>19.5% (17)</td>
<td>0.80</td>
<td>87</td>
</tr>
</tbody>
</table>

If you use another language, please indicate the relative importance of training in it:

- answered question 97
- skipped question 22

"We don’t believe that ... Joe the programmer should have to deal with parallelism in an explicit way”

Kunle Olukotun

Going parallel should not mean writing low-level parallel code, just as going high-performance need not require assembly language (as in the good old days) compilers should help, even though, this is a difficult task. A programming model that presents parallelism to the programmer in a simple yet powerful way can achieve surprisingly good results with the proper compiler support. While at the same time comfortably outperform what most novice programmers would produce with great effort when forced to exercise direct control.
## Compiler Support

<table>
<thead>
<tr>
<th>Compiler</th>
<th>OOP+Functional</th>
<th>Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Intel</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GNU</td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td>IBM</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Portland Group</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>NAG</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- "X" -> a released version supports all features employed in this talk.
- "O" -> a pre-release version supports all features employed in this talk.
Parallel Functional OOP in Modern Fortran

OOP → Functional → Parallel

Parallel programming (Fortran 2008)
Object-oriented programming (Fortran 2003)
Functional Programming (Fortran 95)

User-defined, purely functional operators

$$u_t = -\frac{\text{grad}.p}{\rho} + \nu(\text{laplacian}.u) - (u\cdot\text{grad}.u)$$

Distributed objects containing coarrays
Program main
  implicit none
integer, parameter :: num_particles=10, num_dimensions=3
integer :: stride
real :: V(num_particles, num_dimensions), response_time(num_particles)
V=1.0.
stride = input_from_file()
do concurrent (particle=1:num_particles:stride)
  V(particle,:) = &
  V(particle,:) - dt*V(particle,:)/response_time(particle)
  !Can also call pure procedures here
end do
end program

- Supporting compiler technologies:
  - Gfortran: SIMD via front-end pragmas
  - Intel: Vectorization (AVX)
Case Study: Morfeus

A coordinate-free PDE solver framework:

```fortran
program main
  use cartesian_tensor_class, only : cartesian_tensor
  use scalar_field_class, only : scalar_field
  implicit none
  type(cartesian_tensor) :: u
  real :: t=0.,dt=0.1,t_final=1.0,nu=0.01
  type(scalar_field) :: initial(3)
  u=cartesian_tensor(initial,rank=1,space_dim=3,covariant=[.true.])
  do while(t<t_final)
    u = u + dt*( nu*(.laplacian.u) - (u.dot(.grad.u)) )
    t = t + dt
  end do
end program
```

\[
\vec{u}^{n+1} = \vec{u}^n + \Delta t \left( \nu \nabla^2 \vec{u}^n - \vec{u}^n \cdot \nabla \vec{u}^n \right)
\]

u = u + dt*( nu*(.laplacian.u) - (u.dot(.grad.u)) )

\[
t = t + dt
\]
Asynchronous Expression Evaluation

\[ \nu \ast (u(1) \% xx() + u(1) \% yy() + u(1) \% zz()) \\
- (u(1) \ast u(1) \% x() + u(2) \ast u(1) \% y() + u(3) \ast u(1) \% z()) \]

Legend
- Pure function
- Impure function
1D Burgers Solver Load Balance
Results: 1D Burgers Equation Weak Scaling

![Graph showing Efficiency vs. Number of Images for Weak Scaling]
Fortran Philosophy

“Communicate properties, not optimizations.”
Conclusions

- Fortran is now a PGAS language with a platform-agnostic, scalable parallel programming model.
- Modern Fortran supports multiple programming paradigms that fully integrate with its PGAS features: array programming, functional programming, object-oriented programming.
- Very broad support for OOP/Functional programming features exists.
- Parallel programming feature support is growing.
- Productivity for beginners is high.
References


Acknowledgements

Karla Morris, Sandia National Laboratories

Hari Radhakrishnan and Stavros Kassinos, University of Cyprus

Magne Haveraaen, University of Bergen

Jim Xia, IBM

Xiaofeng Xu, GM

Sameer Shende, University of Oregon