Addressing Unstructuredness and Hardware/Software Divergence

Henry Weller
OpenCFD Ltd.

23rd October 2012

Outline

Background

- 1986: undergraduate CFD course
- 1988: joined Prof. Gosman's CFD group at Imperial College
- Ambitious European funded project to simulate IC engines
  - complete complex geometry
  - piston motion, inlet and exhaust valves
  - full physics: gas, spray, combustion, pollutants *etc.*
  - $\rightarrow$ SPEED: simulation package for engine evaluation and design

SPEED

- FORTRAN 77 on UNIX workstations
- Machine support: scalar/vector
- FORTRAN 77 is a miserable language for complex code development
- UNIX workstations are an excellent code development platform
- Workstation performance improved at a remarkable rate
- Ported to VAX, IBM 3090, Cray XMP and YMP

Mesh

- Unstructured mesh an absolute necessity
- cell-based indirect addressing (unrolled)

```
      DO 20 IFS=-1,1,2
      DO 20 IFF=1,3,ISTEP
         L=IFS*IFF
         DO 30 IC=1,NC
            NNC=NEIGHB(IC,L)
            WP= WEIGHT(IC,L)
            PHIWF=WP*PHI(IC)+(1.0-WP)*PHI(NNC)
            GRAD(IC,1) = GRAD(IC,1)+ACELL(IC,L,1)*PHIWF
            GRAD(IC,2) = GRAD(IC,2)+ACELL(IC,L,2)*PHIWF
            GRAD(IC,3) = GRAD(IC,3)+ACELL(IC,L,3)*PHIWF
30       CONTINUE
20    CONTINUE
```

  - restricted to one cell shape only: hexahedral
  - More efficient that direct `IJK` addressing; no need to compute addresses

New generation of solvers

- Old methods ADI, Stones etc. not applicable
- Pioneered conjugate-gradient methods for unstructured meshes
- Incomplete Cholesky preconditioner for pressure
- LDU precondition bi-conjugate-gradient for momentum and scalars

Vectorization

- Previous structured-mesh codes Vectorized well
- Unstructured mesh addressing causes problems
  - Need to unroll loops
  - Factor out conditionals
  - Limit dependencies (difficult with indirect addressing)
- Conjugate-gradient vectorizes well
- IC/LDU preconditioning vectorize badly
- “octave” renumbering with the octave length = vector length
  - Vectorizes well
  - Renumbering causes IC/LDU preconditioning to perform badly
  - Less effective than diagonal preconditioning
- Conclusion: abandon vectorization in favour of parallelization

Parallelization: the new beginning

- 1989: multi-cpu workstations and clusters
  - cheap and effective alternatives to old vector machines
- Programming challenge:
  - Complex geometry
  - Unstructured mesh
  - Complex solvers
  - Complex physics
  - Everything parallelizable
- Retrofit parallelization to legacy codes
  - mixed success
- Design code from scratch for extensibility and maintainability

Increasing complexity

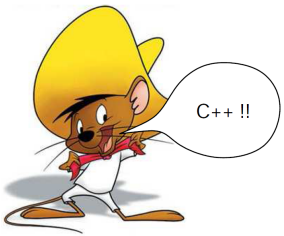
- Brian Kernighan: “Controlling complexity is the essence of computer programming”
- Programming complex geometry and complex physics is extremely challenging
- Need to use the best tools
- Backward-compatibility constrains code development
- Reusing old code limits future code quality

Choose the best abstractions

- Design carefully with a view to extension
- Redesign and rewrite as necessary
- Field algebra and continuum mechanics are powerful tools to model the world
  - Map these into code as closely as possible

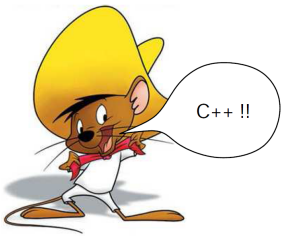
Choose the best programming language

Choose the best programming language



Choose the best programming language

- Introduced to C++ in 1989 – way better than FORTRAN 77!
- Object-orientation is a powerful paradigm for complex code
- Templates (generic programming) is even more powerful
  - Flexible
  - Efficient
  - Field algebra maps to code through templates effectively → FOAM
- C++ great but not perfect
  - templates were a bolt-on but good proof of concept
  - C++ unnecessarily difficult to use



FOAM (Field Operation and Manipulation)

- Start designing new parallel, flexible, extensible and intuitive CFD framework in C++
- Secured funding for a Mico computing surface
- Failed to secure funding for FOAM
- Developed FOAM in my own time
- FOAM released open-source as OpenFOAM by OpenCFD in 2004

Structure

- Fundamentally parallel
- General extensible framework
- Unstructured mesh
- Indirect addressing
- Support for arbitrary shaped cells in 3D
- Face-based addressing
  - Loop over all faces in mesh
  - For each face lookup “owner” and “neighbour” cells
  - Simpler and more flexible than cell+direction/direction+cell-based
  - Intuitive for Gauss integrals

```
forAll(owner, facei)
{
    ivf[owner[facei]] += issf[facei];
    ivf[neighbour[facei]] -= issf[facei];
}
```

  - Less intuitive for ICCG, Gauss-Seidel, *etc.*
  - Requires special ordering of the faces
- ICCG and ILUBiCG solvers
- Added AMG and GAMG solvers
  - Much more efficient than ICCG on large complex problems
  - but increased parallel communication per iteration

Success!

- Can simulate a very wide range of physics effectively
  - Aerospace: hypersonic flight (extended hydrodynamics and DSMC)
  - Automotive: aerodynamics (F1), flow-induced noise, injection systems, catalytic converters, bearings, engines...
  - Chemical/biomedical: mixing, separating, fluidized beds, cardiovascular devices (FSI)
  - HVAC: automotive/aero systems, icing...
  - Marine: tank sloshing, drag, cavitation (propellers, hydrofoils)
  - Micro/nanofluidics: ink-jet printing, coatings, molecular dynamics
  - Misc: baking, crystal growth, climatic flows, insect flight, financial derivatives...
  - Oil and gas: separation, fuel containment, risk assessment
  - Nuclear power: heat transfer, thermal fatigue
  - Power generation: fluidised beds
  - Pumps: piston lubrication
  - Safety: fires, explosions, pollution dispersion...
- OpenCFD continue to develop and support OpenFOAM for thousands of users (over 1million downloads so far)

Increasing demands

- F1 car: 50million cells on workstation in 24hrs
- LES/DES, multiphase, phase-change, reaction, rheology, particle mechanics *etc.* require more and more compute power
- CPUs increase in performance
- Cores per socket increases
- Network performance increases
- @BUT. . . @

CPU vs Memory

- Memory performance didn't keep-up with CPU performance
- Memory latency and bandwidth now limiting factor
- Problem for exploiting new technology: multi-core, GPU, MIC etc.
  - Vector machines in disguise
  - Need to localise memory access
  - Need to vectorize
  - Need to micro-scale parallelization
  - @Unstructured mesh indirect addressing a problem@
- Sandybridge: only 30% of 100% speedup why?
  - 4 memory channels rather than 3

Choices

- Complex geometry, complex physics: no choice
- Unstructured mesh: Is there a choice?
  - block-structured, direct addressing within block?
  - hierarchical, direct addressing within level?
  - Immersed-boundary?
- Implicit/Semi-implicit: Is there a choice?
  - for compressible flow: explicit?
  - for incompressible flow: artificial compressibility?

Should we wind-back the clock?

- Explicit
- Block-structured
- Take advantage of new technology: multi-core, GPU, MIC etc.
- Does this really avoid the memory bottleneck?

Should we wind-back the clock?

- Explicit
- Block-structured
- Take advantage of new technology: multi-core, GPU, MIC etc.
- Does this really avoid the memory bottleneck?
- No, only moderates the problem
- Probably advantageous only for some classes of problems

New memory technology

- Hybrid Memory Cube (Samsung, Micron, IBM) http://hybridmemorycube.org/
  - HMC prototype modules deliver ~128GB/sec bandwidth vs 12.8GB/sec for DDR3
  - Claimed reduction in latency (no numbers yet)

New memory technology

- Hybrid Memory Cube (Samsung, Micron, IBM) http://hybridmemorycube.org/
  - HMC prototype modules deliver ~128GB/sec bandwidth vs 12.8GB/sec for DDR3
  - Claimed reduction in latency (no numbers yet)
- Intel's Teraflops Research Chip (Polaris) 2007
  - http://en.wikipedia.org/wiki/Teraflops_Research_Chip
  - 80 relatively simple and low power cores
  - SRAM chip ("Freya") stacked directly underneath the cores
  - @Scalable: removes memory bottle-neck@
  - Power reduced from 25pJ/bit to 1pJ/bit
  - Bandwidth 1TB/s
  - Problem with thermal stress in the memory layer
  - Expensive to develop

Not enough money in HPC but . . . .

Not enough money in HPC but . . . .

- . . . We are in luck:
  - the things that make computers slow also make them consume more power

Not enough money in HPC but . . . .

- . . . We are in luck:
  - the things that make computers slow also make them consume more power
- Piggy-back on developments for other more lucrative industries
  - Gaming: multi-core, GPU
  - @Mobile devices: low power consumption@

Not enough money in HPC but . . . .

- . . . We are in luck:
  - the things that make computers slow also make them consume more power
- Piggy-back on developments for other more lucrative industries
  - Gaming: multi-core, GPU
  - @Mobile devices: low power consumption@
- Mobile industry may fund 3D and/or stacked memory

Not enough money in HPC but . . . .

- . . . We are in luck:
  - the things that make computers slow also make them consume more power
- Piggy-back on developments for other more lucrative industries
  - Gaming: multi-core, GPU
  - @Mobile devices: low power consumption@
- Mobile industry may fund 3D and/or stacked memory
- Need to encourage demand for mobile gaming!
- Possible future for HPC: arrays of low-power cores with stacked memory?
- Will Intel resurrect Polaris or something similar?

Conclusion: @No U-turn . . . yet@

- Decades of research and algorithm development $\rightarrow$
  - efficient, effective, flexible, general and extensible techniques
- Possible that memory developments will resolve the current bottle-neck
- Software development slower than hardware development *e.g.*
  - OpenFOAM has taken 23years so far (partly due to lack of funding)
- Don't give up on unstructured mesh, semi-implicit algorithms . . . yet.