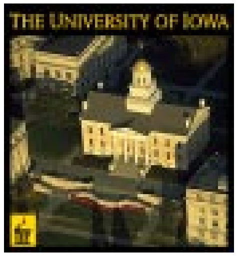


Development, Performance and Scalability, and Portability of a MPI-based version of CFDSHIP-IOWA: Results of a NAVO PET Tiger-Team Collaboration

Eric G. Paterson
Iowa Institute of Hydraulic Research
The University of Iowa
Iowa City, IA

Robert S. Sinkovits
San Diego Supercomputer Center
San Diego, CA

9th DoD HPCMP Users Group Conference
Monterey, CA
8-10 June 1998

Outline

- Introduction
- Goals/Requirements
- CFDSHIP-IOWA
  - algorithm
  - data structure and code architecture
  - approach to parallelization and load balancing
  - portability
- Benchmark example
- Concluding Remarks

Introduction

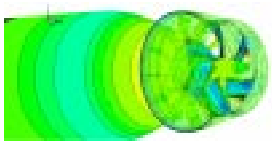
- Collaboration was a NAVO PET Tiger Team project
  - The Tiger Team mission
    - identify Challenge-class and/or CHSSI researchers in the NAVO Computational Technology Areas
    - work with them to achieve breakthrough results by combining HPC skills and experience with scientific expertise
- Scope of collaboration
  - accelerate code development work already underway so as to achieve a stable, production-level code by project completion
  - transition HPC knowledge from SDSC to IIHR
  - transition to HPCMP parallel platforms

Introduction

- CFDSHIP-IOWA
  - general-purpose unsteady incompressible Reynolds-averaged Navier-Stokes CFD code
  - developed for application to Naval hydrodynamics
  - DoD HPCMP Challenge project (Rood, 1999; Rood, 1998; and Rood, 1997): unsteady RANS simulations for maneuvering ships (6DOF) in waves
  - vertical integration: code transitioned to industry, Navy laboratories, and other universities
    - HPC impact on acquisition: DD-21 and beyond

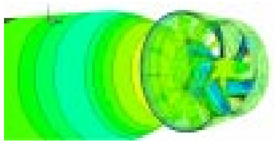
Goals/Requirements

- Parallelization achieved through the use of MPI and a coarse-grain parallel multi-block scheme.
- Static load balancing achieved through domain decomposition
- Parallel execution ≡ serial execution
- Portable: run on the range of HPCMP computers (SGI Origin 2000, CRAY T90, CRAY T3E).
- The final code should be
  - readable, well structured, and easy to maintain
  - user community should be able to add new model subroutines with a minimum of difficulty



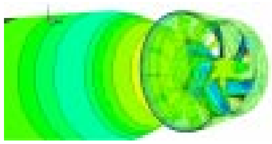
CFDSHIP-IOWA

- Governing equations
  - 3D unsteady incompressible RANS equations
    - several base coordinate systems
      - Cartesian, cylindrical polar, non-inertial cylindrical polar
  - Reynolds-stress closure
    - standard linear stress-strain relationship and a variety of eddy-viscosity models
      - algebraic Baldwin-Lomax
      - 2-equation k-ω and k-ε models.
  - Air-water free-surface interface
    - Kinematic free surface boundary condition: hyperbolic wave equation
    - Grid dynamically conforms to both ship hull and wave field

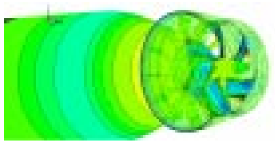


CFDSHIP-IOWA

- Algorithm/scheme
  - fully implicit PISO algorithm
  - higher-order finite-difference (FD) operators
    - convective terms: general 5-point stencil (user-specified accuracy: 1st- to 4th-order)
    - viscous terms: 2^{nd} order central FD
    - temporal terms: 1^{st} or 2^{nd} order backward FD
  - pressure Poisson equation
    - projection of velocity into a divergence-free field at time level (*n*).
    - collocated grid approach is used: oscillations prevented by taking a linear combination of full- and half-cell operators
  - Algebraic equations are solved using line-ADI and a penta-diagonal algorithm



- Data structure & Code Architecture
  - designed for
    - scalable parallel multi-block implementation
    - ease of 2nd-party model subroutine development
  - To achieve this
    - all variables passed to subroutines through argument lists
    - `COMMON` block usage is restricted to global/run-time parameters
    - arrays are 1D at the main program level
    - problem dimensions and block pointers are in ‘`cfd_ship.h`’
    - data for each block passed to subroutines separately
    - subroutine arrays are 3D and have dimensions dynamically set to size of given block
    - subroutine authors can write code as if CFDSHIP-IOWA was a single-block serial program.



CFDSHIP-IOWA

<u>Main Program</u>

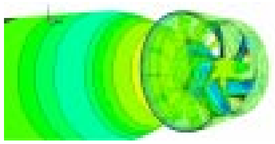
```
     Implicit none
     integer imax(nmesh),jmax(nmesh),kmax(nmesh)
     integer first(nmesh)
     real*8 u(ntot)

#IFDEF PARALLEL
     call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
     m=myid+1
#ENDIF

#IFDEF SERIAL
     do m=1,nmesh
       locn=first(m)
       call example(imax(m),jmax(m),kmax(m),u(locn))
     enddo
#ENDIF
#IFDEF PARALLEL
     locn=1
     call example(imax(m),jmax(m),kmax(m),u(locn))
#ENDIF
```

<u>Subroutine Example</u>

```
     subroutine example(imax,jmax,kmax,u)
     implicit none
     integer imax,jmax,kmax
     real*8 u(imax,jmax,kmax)
        ...
        ...
        ...
     return
     end
```



CFDSHIP-IOWA

- Approach to Parallelism
  - data structure easily permits parallel multi-block implementation via MPI and the single-program multiple data (SPMD) paradigm.
  - Except for the input and output routines, communication between processors occurs only at the boundary condition subroutines
    - can take the form of either patched- or overset-grid multi-block boundary conditions.
  - Processor 0, through the use of branching statements, handles control of input and output
  - all of the MPI coding is contained in either the main code ‘cfdship_main.F’ or a module named ‘cfdship_mpsr.F’, the latter of which is about 3300 lines of code.

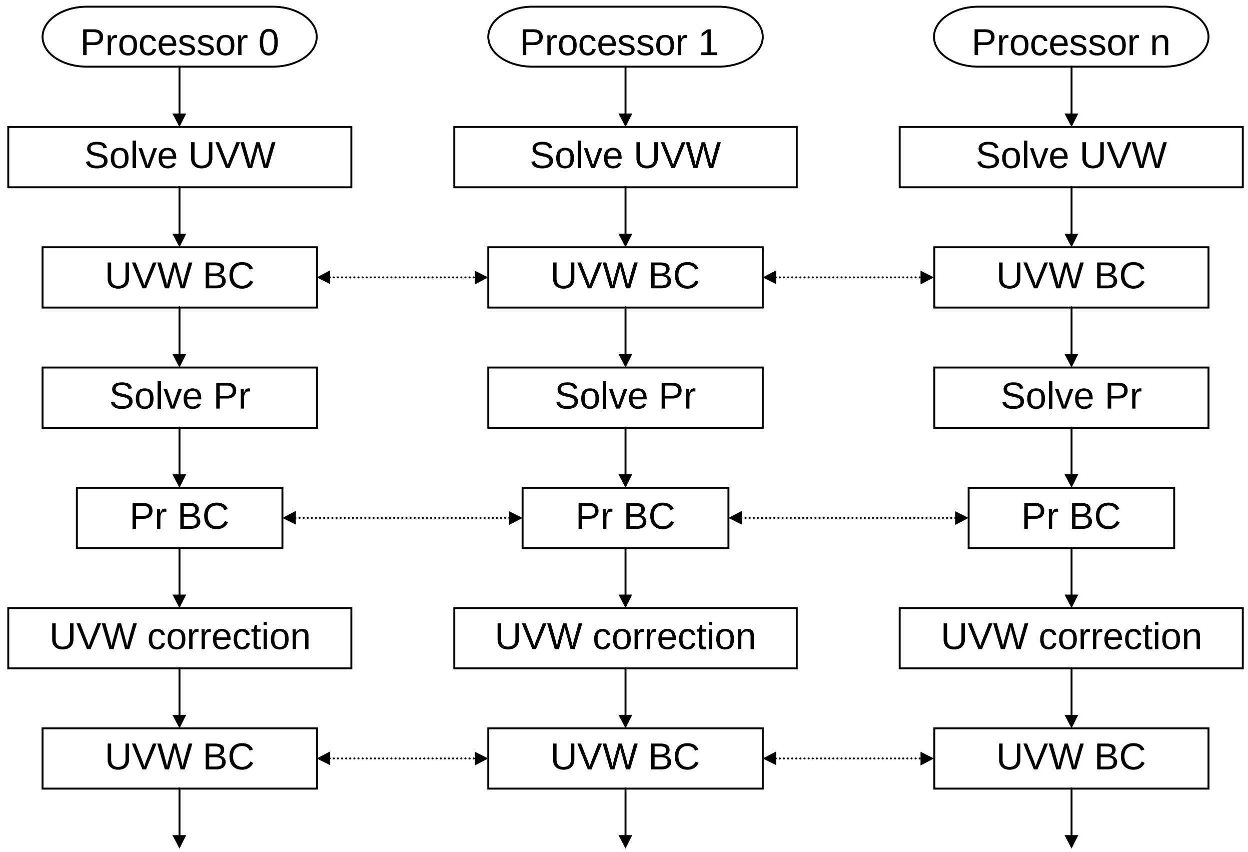
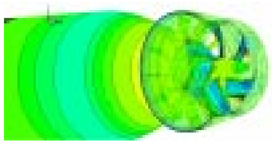
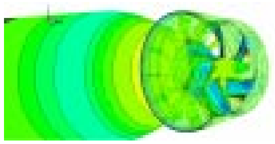


Figure 2. SPMD Architecture

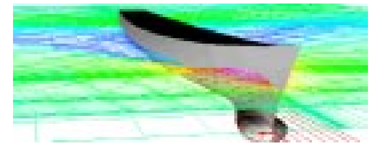


CFDSHIP-IOWA

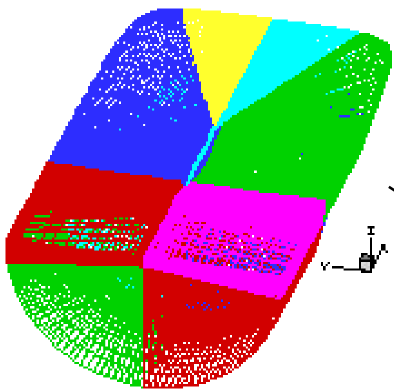
- Load balancing is accomplished via domain decomposition
- Places burden on user.
  - Uniform domain decomposition is often difficult to achieve, especially for complex geometries.
  - include file and boundary-condition input data must be prepared for N-block systems
    - development of preprocessor, ‘decomp.f’
  - Visualization and post-processing becomes more time consuming since, at this time, the decomposed grid system is not re-assembled back into its original state.

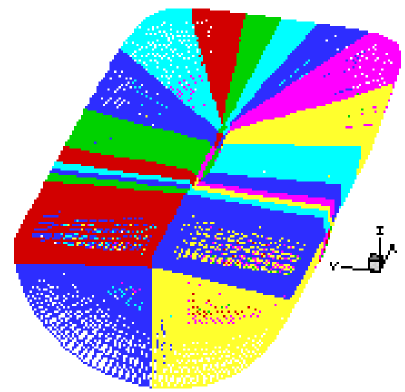


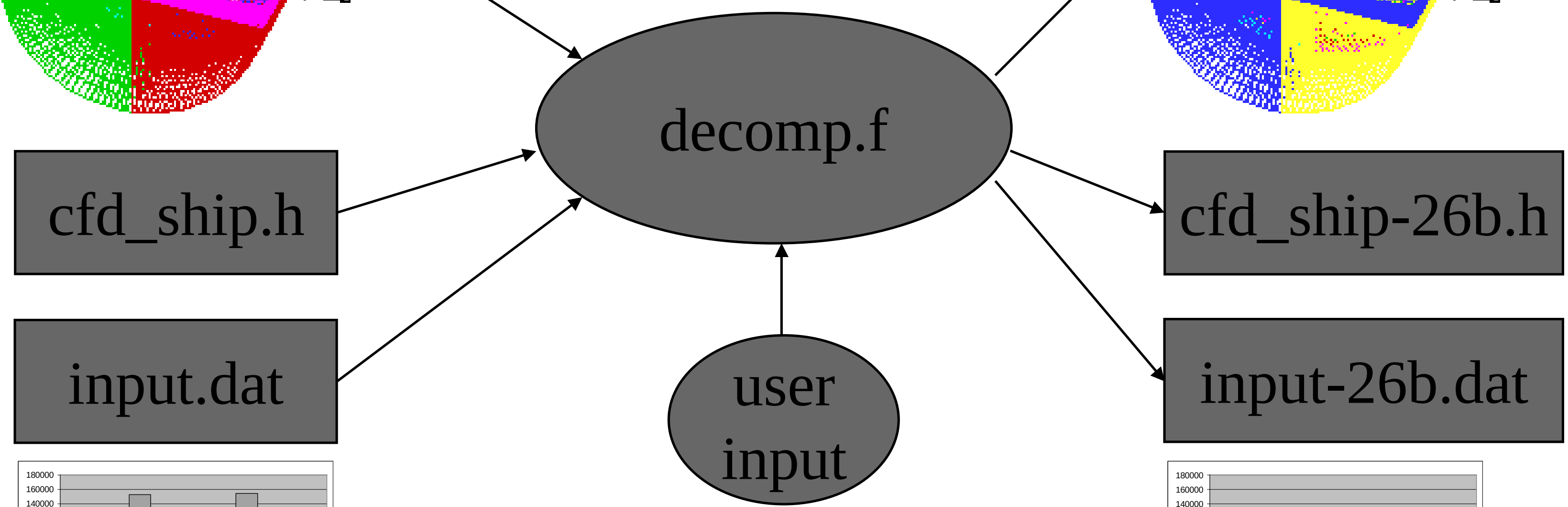
CFDSHIP-IOWA

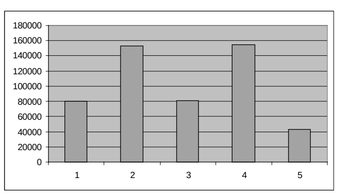


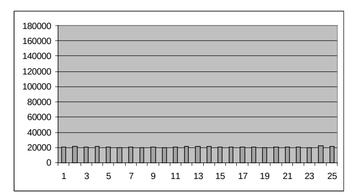
Domain Decomposition &
Load Balancing
- Model 5415 -

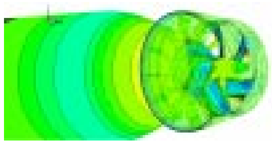












CFDSHIP-IOWA

- Portability
  - achieved through the use of
    - MPI
    - the C preprocessor CPP
      - used to build either parallel or serial versions from a single source code
      - MPI- and serial-specific code is isolated through use of `'#IFDEF SERIAL'`, `'#IFDEF PARALLEL'`, and `'#ENDIF'` `CPP` directives
    - UNIX make utility
      - platform specific versions by invoking correct compiler options and `CPP` directives

Machine	Version	make argument
SGI Origin 2000	Serial	O2K
SGI Origin 2000	Parallel	O2K_MPI
SGI Power Challenge Array	Serial	PCA
SGI Power Challenge Array	Parallel	PCA_MPI
CRAY T3E	Serial	T3E
CRAY T3E	Parallel	T3E_MPI
CRAY T90	Serial	T90
DEC Alpha	Serial	DEC
HP workstation	Serial	HP

Example

- Wigley Hull
  - frequently used benchmark in the computational ship hydrodynamics community
    - simple, analytical geometry (canoe body)
      - single-block grid generation
    - breadth of experimental data
  - one of the training example problems found on the CFDSHIP-IOWA web site: http://www.iihr.uiowa.edu/~cfdship/.
  - used to demonstrate the parallel performance on both the SGI Origin 2000 and the Cray T3E

Example

- Grids
  - Two single-block grids were generated
    - 105x61x30
    - 209x121x59
  - decomp.f tool was used to decompose the grid system

Table 2. Block size (ntot) and total grid point (ntot_sum) variation vs. number of blocks.

# of blocks	Grid 1 ntot	Grid 1 ntot_sum	Grid 2 ntot	Grid 2 ntot_sum
1	192,150	192,150	1,492,051	1,492,051
2	96,960	193,980	749,595	1,499,190
4	49,410	197,640	378,367	1,513,468
8	25,620	204,960	192,753	1,542,024
16	13,020	208,320	97,173	1,554,768
32	6,720	215,040	49,383	1,580,256
64	3,584	222,208	25,110	1,607,040

Example

- Grids
  - decomposition was first made in ξ=constant planes (7 cuts), then in η=constant planes (3 cuts), and then in ζ=constant planes (1 cut)

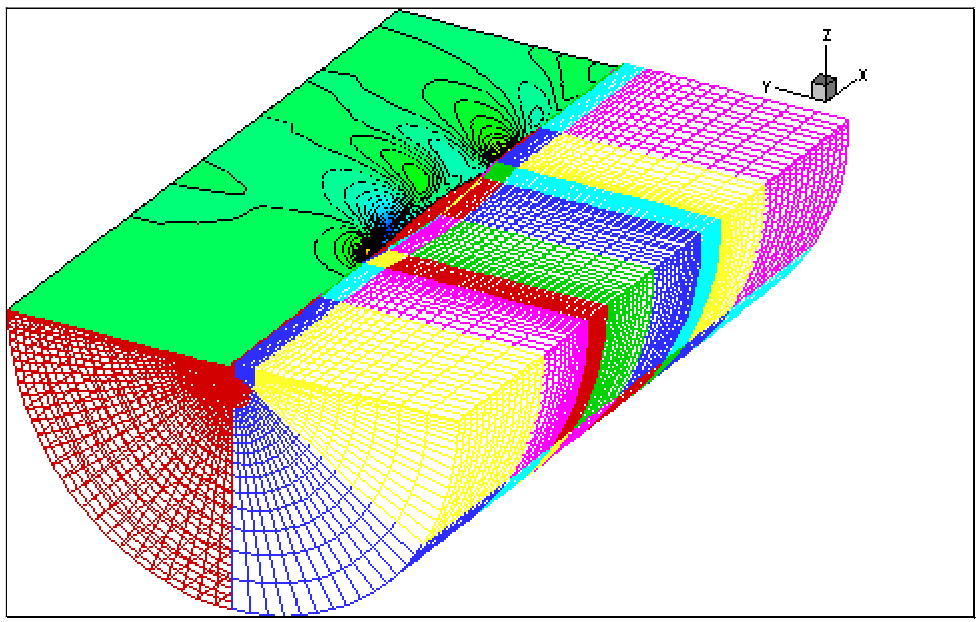
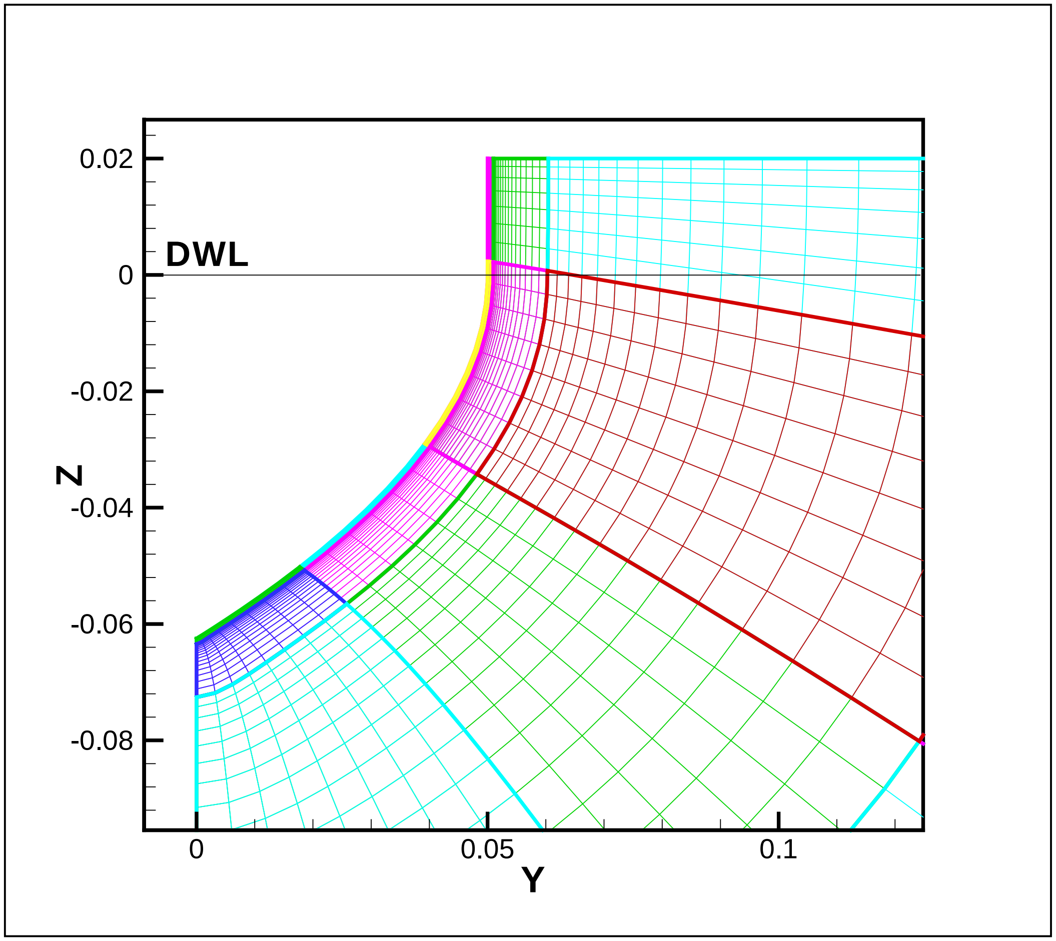


Figure 3. 64-block grid system for Wigley Hull

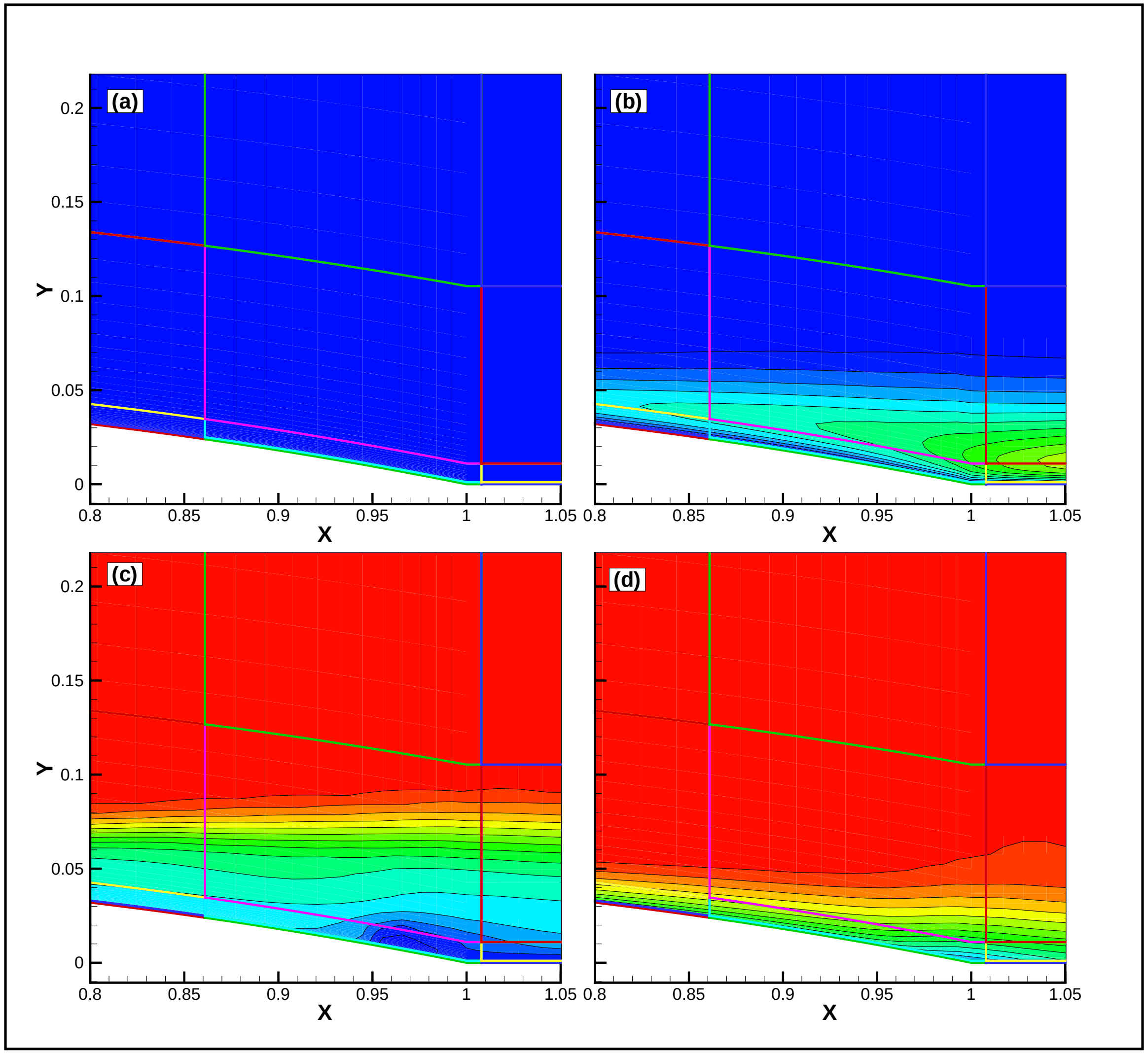
Example

- Certain constraints are imposed on domain decomposition.
  - For free-surface calculations, decomposition must be undertaken with care due to the dynamic free-surface conforming grid
  - For turbulent flow, it is recommended that the Baldwin-Lomax (BL) turbulence model be used with caution and that, in general, the k-ω models be used as the standard approach

Example



Example



Example

- Performance
  - Wall-clock time measured using calls to MPI_WTIME
    - for entire program
    - for the main portion of the code, which excludes input, initialization, and output.
  - Timings show that 99.99+% of the CPU time is spent in the main portion
  - Timings were made during normal operation of the systems, i.e., dedicated resources were not obtained.
    - NAVO O2K was very heavily loaded and timings were not repeatable.
  - Speed-up on both grids shown on following page

Example

- Performance

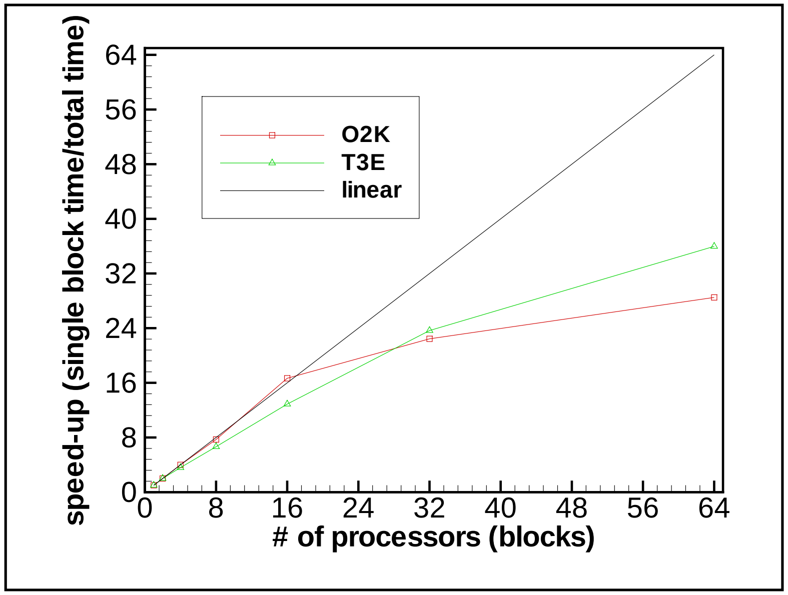


Figure 6. Parallel speed-up for Grid 1

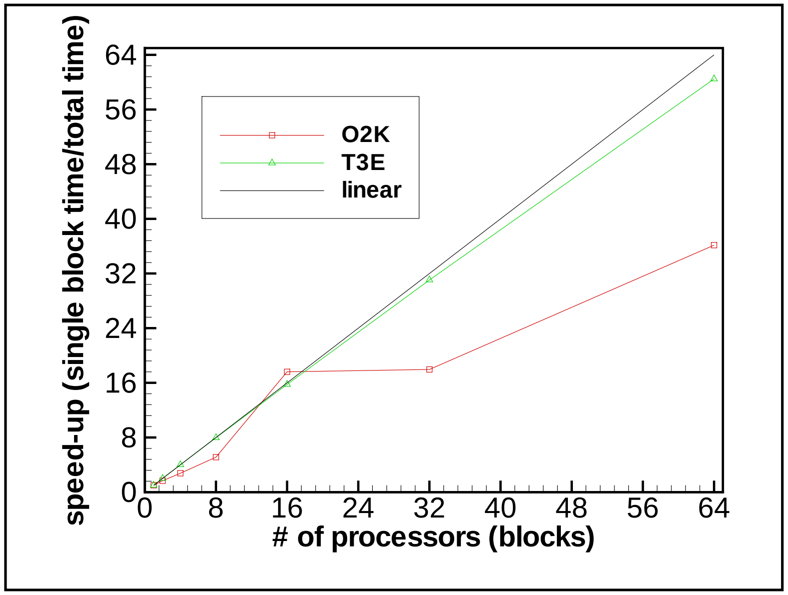


Figure 7. Parallel speed-up for Grid 2

Concluding Remarks

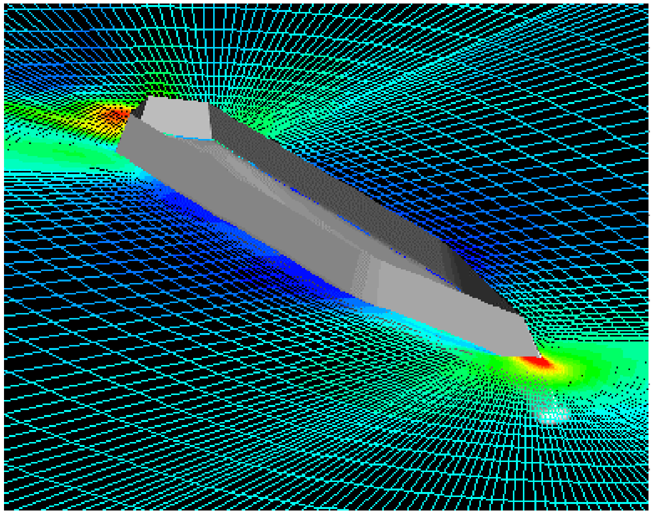
- This collaboration was very successful: the established goals and requirements were all met
  – CFDSHIP-IOWA is now a high-performance portable CFD code
  – IIHR staff and students have fully transitioned to HPCMP platforms such as the O2K and the T3E
  – Performance is good and the scalability of memory provides the capability for solving problems significantly larger than possible on the vector machines such as CRAY T90

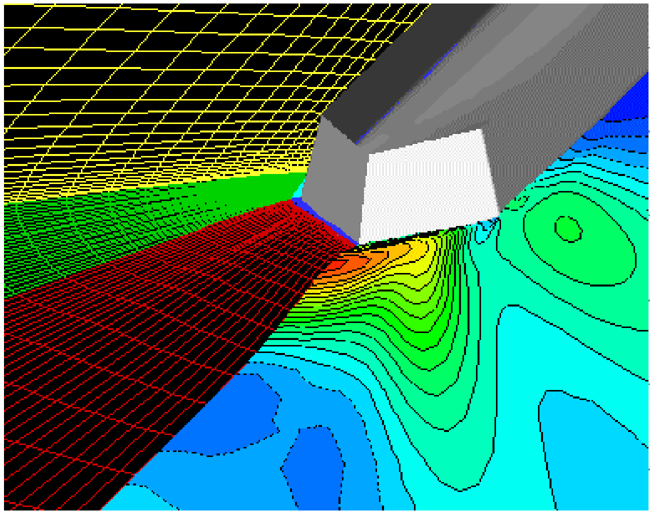
Concluding Remarks

- Collaboration has been very timely
  - CFDSHIP-IOWA is being applied to revolutionary new hull geometries for the stealth ship DD-21, the Navy's new land attack destroyer

Concluding Remarks

- Performance and scalability is critical
  - capability for 15-20 Million grid points on 128-processor O2K
  - short turnaround is critical for practical design and analysis
- Competing shipyards for the DD-21 award have asked to receive the products from this collaboration and the related challenge project





Concluding Remarks

- Future work is already underway
  - The most pressing need is to develop a more flexible approach to domain decomposition
  - In the present scheme, there is a one-to-one mapping between blocks and processors
    - requires great effort be expended in the grid generation stage to ensure an even distribution of work
    - certain restrictions exist on decomposition
  - A solution to this problem is *multi-level parallelism* and *dynamic load balancing* using MPI and OpenMP together in a coarse-grain/fine-grain approach.