Software for Solving Elliptic PDEs With Parallel Adaptive Multi-level Methods

William F. Mitchell

Mathematical and Computational Sciences Division
National Institute of Standards and Technology
Gaithersburg, MD

June 3, 2004
The Changing Face of Mathematical Software
PHAML

- Parallel Hierarchical Adaptive Multi-Level
- Fortran 90 program
- 2D Elliptic PDE solver (BVP and eigenvalue)
- Adaptive finite elements with multigrid
- Message passing parallelism: MPI or PVM
- Optional: Zoltan, OpenGL, ARPACK, PetSC, hypre, MUMPS, SuperLU
Outline

★ Computational Setting
★ Numerical Methods
★ Software Design
★ User Interface
★ Graphics
★ Conclusion
Elliptic Boundary Value Problems

\[- \frac{\partial}{\partial x} p \frac{\partial u}{\partial x} - \frac{\partial}{\partial y} q \frac{\partial u}{\partial y} + ru = f \quad \text{in} \quad \Omega \subset \mathbb{R}^2\]

\[u = g_1 \quad \text{on} \quad \partial \Omega_1\]

\[\frac{\partial u}{\partial n} + cu = g_2 \quad \text{on} \quad \partial \Omega_2\]

\[p, q, r, f, c, g_1 \text{ and } g_2 \text{ functions of } (x, y)\]

\[\partial \Omega_1 \cup \partial \Omega_2 = \partial \Omega\]

\[\Omega \text{ polygonal}\]

Also: time dependent, nonlinear, systems
Elliptic Eigenvalue Problems

\[-\frac{\partial}{\partial x} p \frac{\partial u}{\partial x} - \frac{\partial}{\partial y} q \frac{\partial u}{\partial y} + ru = \lambda u \text{ in } \Omega \subset \mathbb{R}^2\]

\[u = 0 \text{ on } \partial\Omega_1\]

\[\frac{\partial u}{\partial n} = 0 \text{ on } \partial\Omega_2\]

\(p, q, r \text{ functions of } (x, y)\)

\(\partial\Omega_1 \cup \partial\Omega_2 = \partial\Omega\)

\(\Omega \text{ polygonal, generally a truncation of an infinite domain}\)
Computational Environment

- Network of workstations or PCs
  - modest number of processors (10's)
  - distributed memory parallelism with message passing
  - now a common parallel computer in resource-limited labs
- vs. expensive massively parallel machines
  - smaller bandwidth
  - much higher latency
  - need parallel algorithms with infrequent messages
Numerical Methods

- Parallelization of methods in MGGHAT
- Standard finite elements, linear, triangles
- Newest node bisection adaptive refinement
- Hierarchical basis multigrid
- Refinement–tree based grid partitioning
- Full domain partition
Adaptive Multilevel Algorithm

start with very coarse mesh
repeat
  if the load is too far out of balance
    repartition grid
    redistribute data
  endif
  adaptive mesh refinement
  multigrid cycles
until termination criterion is met
Full Domain Partition

• Each processor gets the triangles from a partition
• Each processor receives additional shadow data
  • just enough to cover the full domain
  • taken from coarser refinement levels
  • $O(\sqrt{N})$ extra data
Full Domain Partition Example
Full Domain Partition Example
Grid Partitioning

- Goal: balance the load, minimize communication
- Must be very fast
- Must work on distributed data
- Zoltan library (Devine et al., Sandia)
  - recursive coordinate and inertial bisection
  - space filling curve methods
  - refinement-tree based method
  - ParMetis, Jostle
Refinement-tree Based Method

- Refinement tree
  - nodes correspond to elements
  - children of a node are elements created by refinement
- Label the leaves with a weight
- Traverse the tree to compute subtree weights
  - the only communication occurs during this step
- Traverse the tree to assign subtrees to a partition until the partition contains \((total\ weight)/p\)
- Achieves perfect balance, contiguous partitions
Adaptive Refinement

- Goal is to concentrate the effort where it does the most good
- Estimate the error on each element and refine those with the largest error estimate
- Many methods for:
  - error estimate
  - element refinement
  - maintaining compatibility
  - when to quit
Parallel Adaptive Refinement

- Refine each partition independently, but only refine *owned* triangles with large error estimate and triangles for compatibility
- Send a list of unowned triangles that were refined to the other processors
- Refine owned triangles that other processors refined
- Enforce overlap requirements
Multigrid

- Use a sequence of grids to solve in O(N) operations
  - Smooth error on current grid (e.g. Gauss-Siedel)
    - Reduces high frequency components of error
  - Define and solve a problem on a coarser grid
    - Lower frequency components are high frequency here
    - Recursively apply procedure
  - Use coarse grid solution to correct fine grid solution
  - Smooth again on fine grid

- For a large class of PDEs, reduces the error by a contraction factor independent of N
Hierarchical Multigrid

\[ A_c x_c = b_c - A_{cf}^T x_f \]
Parallel Hierarchical Multigrid

\[ A_c x_c = b_c - A_{cf}^T x_f - A_{cf}^T x_f - A_{cf}^T x_f - A_{cf}^T x_f \]
Parallel Hierarchical Multigrid

downward part of cycle
send new solution and $A^T_{cf} x_f$ to other processors
solve on coarsest grid
upward part of cycle
send new solution to other processors
Parallel Efficiency

Scaled Problem Size

132,000 nodes per processor

Efficiency

Refinement  50%
Multigrid    75%
Overall     66%
Fortran 90 Modules

- Contain variables, constants, type definitions, etc.
- Entities can be public or private
- Use for
  - Global data
  - Library interface definition
  - Data encapsulation
  - Many other uses
Data Encapsulation

- Public type with private internals
- Operations on that type
- PHAML has two forms of hash_key

```plaintext
public hash_key, hash_insert, hash_remove

type hash_key
  private
  integer :: key
end type hash_key
```
Primary PHAML Modules

- phaml
- grid, grid_type
- linear_system and subordinates
- load_balance
- message_passing
- hash, sort
- graphics
- global
- interfaces to other software packages
Data Structures

type phaml_solution_type
  private
  type(grid_type) :: grid
  type(proc_info) :: procs
  integer :: outunit, errunit, pde_id
  character(len=HOSTLEN) :: graphics_host
  logical :: i_draw_grid, master_draws_grid, &
             still_sequential
end type phaml_solution_type
Data Structures

type grid_type
    type(element_type), pointer :: element(:)
type(node_type), pointer :: node(:)
type(hash_table) :: elem_hash, node_hash
integer :: next_free_elem, next_free_node
integer, pointer :: head_level_elem(:), &
    head_level_node(:)
integer :: partition
integer :: nelem, nelem_leaf, nelem_leaf_own, &
    nnode, nnode_own, nlev
end type grid_type
Data Structures

type node_type
    type(hash_key) :: gid
    type(point) :: coord
    real :: solution
    integer :: type, assoc_elem, next, previous
end type node_type
User Interface

• User written external subroutines
  • pdecoefs, bconds, iconds, trues, true_energies_sq, init_grid, integral_kernel

• Public PHAML subroutines
  • create, destroy, solve_pde, evaluate, query, integrate, connect, store, restore, popen, pclose

• Main program that uses the PHAML library
User Written Subroutines

subroutine pdecoefs(x,y,cxx,cyy,c,rs)

! pde is
! -( cxx(x,y)*u ) -( cyy(x,y)*u ) +c(x,y)*u = rs(x,y)
! x x y y

real, intent(in) :: x, y
real, intent(out) :: cxx(:,:),cyy(:,:),c(:,:),rs(:)

cxx(1,1) = 1.0
ccyy(1,1) = 1.0
c(1,1) = 0.0
rs(1) = x*x + y*y

end subroutine pdecoefs
Public PHAML Subroutines

- Operate on `phaml_solution_type` variables
- Optional arguments
  - Simplify calling sequences
  - Reasonable defaults for missing arguments
  - Some arguments useful only to experts
- Help upward compatibility
Arguments for solve_pde

```fortran
subroutine solve_pde(phaml_solution, iterm, &
  max_elem, max_node, max_lev, max_refsolvloop, &
  init_form, comm_freq, partition_size, eq_type, &
  print_grid_when, print_grid_who, print_error_when, &
  print_error_who, print_time_when, print_time_who, &
  print_eval_when, print_eval_who, &
  print_header_who, print_trailer_who, clocks, &
  draw_grid_when, draw_reftree_when, pause_after_draw, &
  pause_after_phases, pause_at_start, pause_at_end, &
  uniform, overlap, sequential_node, inc_factor, &
  error_estimator, refterm, derefine, &
  partition_method, predictive, &
  solver, preconditioner, mg_cycles, mg_prerelax, &
  mg_postrelax, iterations, ignore_quad_err, &
  final_solves, final_mg_cycles, &
  num_eval, lambda0)
```
Example main Program

program user_main_example
use phaml

type(phaml_solution_type) :: pde

call create(pde, draw_grid_who = MASTER)
call solve_pde(pde,
               &
               max_node = 20000,
               &
               draw_grid_when = PHASES,
               &
               partition_method = ZOLTAN_RCB,
               &
               mg_cycles = 2)
call destroy(pde)
end program
Parallelism

- Distributed memory computers
  - DM parallel computers, e.g. SP2
  - Clusters
- Message passing
  - MPI, PVM
- Parallel modes
  - Spawning: master/slave/graphics
  - Spawnless: SPMD
  - Sequential
Parallelism

- Four versions of module `message_passing`
- All communication goes through this module
- Parallelism is hidden
  - User provides main program only for master
- Consider slaves/graphics as part of a `phaml_solution_type` object
  - Parallel config hidden in component `proc_info`
  - Processes spawned by subroutine create
  - Processes killed by subroutine destroy
Graphics
Conclusion

- PHAML is a new elliptic PDE solver
- Parallel sequel to MGGHAT
- Adaptive refinement, multigrid, message passing
- Fortran 90 improves code quality, user interface
- Tested on several unixes, compilers
- PHAML is in the public domain
  - http://math.nist.gov/phaml
  - william.mitchell@nist.gov