PROPHET Internals

- PDE description
  - externals
- Database
  - externals
  - internals
- Grid datastructures
- PDE description
  - internals
- Solution flowchart
- Matrix structure

User level
- Input parser 3.3K
- Graphics 2.2K

Modules
- Solve 0.2K
- Grid 0.9K
- Field 0.1K
- Bias 0.6K
- ...

PDE level
- Assembly Control 9.2K
- Discretization ("geoterm") 8.2K
- Models ("phyterms") 3.6K

Libraries
- Database 2.0K
- Vexpr 1.4K
- Grid d/s + routines 4.1K
- Linear Solver...
System decomposition

- Problem may be composed of several blocks of PDE/ODEs

Loosely coupled

- Blocks may be sparse

Tightly coupled
Differential Equation Specification

- 2 level decomposition
  - PDE is a sum of terms
  - Each term is a combination of a geometrical and a physical operator

\[
PDE = G_1 P_1 + G_2 P_2 + G_3 P_3 + \ldots
\]

- Geometrical operator

\[
\nabla \times \nabla \cdot \frac{\partial}{\partial t}
\]

- Physical operator

\[
F_A = f(A, \nabla A, X, \nabla X)
\]
Same equations, different meshes

• Goal is to separate geometry information from physical information
  ⇒ Model developer should not need to know about grid or discretization

Irregular mesh

Regular mesh
Decomposition of Defect Diffusion

\[ \nabla \cdot (D_I \nabla I) \quad k(IV - V^*) \quad \frac{\partial I}{\partial t} = 0 \]

\[ \nabla \cdot (D_V \nabla V) \quad k(IV - V^*) \quad \frac{\partial V}{\partial t} = 0 \]

2 Laplacians 1 binary recombination 2 transients
Description in database

math.defects = (list) {
    transient = (list) {
        order = (string) "int,vac";
        nterm = (int) 3;
        term0 = (list) {
            geoterm = (string) "box-laplacian";
            phyterm = (string) "lapflux";
            sol = (string) "int";
            dep = (string) "int";
            dectype = (string) "DIAG:DIAG=grad";
        };
        term1 = (list) {
            geoterm = (string) "box-laplacian";
            phyterm = (string) "lapflux";
            sol = (string) "vac";
            dep = (string) "vac";
            dectype = (string) "DIAG:DIAG=grad";
        };
        term2 = (list) {
            geoterm = (string) "diagonalweight";
            phyterm = (string) "bulkrecombination";
            sol = (string) "int,vac";
            dep = (string) "int,vac";
            dectype = (string) "ALL:ALL=conc";
        };
        term3 = (list) {
            geoterm = (string) "interface";
            phyterm = (string) "surfacerecombination";
            sol = (string) "int,vac";
            dep = (string) "int,vac";
            dectype = (string) "DIAG:DIAG=conc";
        };
        maxNewton = (int) 15;
    };
};
Discretized operators

- Divergence operator
  - finite element
  - finite difference

- Upwinding operator

- Nodal weighting operator
  - lumped mass matrix
  - consistent mass matrix

- Interface weighting operator
  - lumped mass matrix
Elements

• $C^k_0$ finite elements

• Shape functions for
  - intervals 1D
  - triangles 2D
  - quads 2D
  - tetrahedra 3D
  - bricks 3D
  - pyramids 3D
  - prisms 3D
Prefabricated physical operators

The following physical operators, among others, are predefined:

<table>
<thead>
<tr>
<th>Associated with divergence operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>lapflux</td>
</tr>
<tr>
<td>equilflux</td>
</tr>
<tr>
<td>drift</td>
</tr>
<tr>
<td>updrift</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Associated with nodal operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>two2one</td>
</tr>
<tr>
<td>poissonflux</td>
</tr>
<tr>
<td>set_active</td>
</tr>
<tr>
<td>elim_carrier</td>
</tr>
<tr>
<td>cluster</td>
</tr>
<tr>
<td>odefunc</td>
</tr>
<tr>
<td>expdecay</td>
</tr>
<tr>
<td>integrate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Associated with the interface operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>segregation</td>
</tr>
<tr>
<td>radiation</td>
</tr>
<tr>
<td>odesurf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Associated with the dirichlet operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>default.dirichlet</td>
</tr>
<tr>
<td>device.dirichlet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod, divide, scale</td>
</tr>
</tbody>
</table>
Writing new modules

- Creating new operators, particularly reactions, is easy
- All operators get a standard list of arguments
- Subroutine must calculate the reaction and its derivatives
- Example: clustering

\[ \frac{\partial l}{\partial t} = (k_{cr} C - k_{cf} IC) + \text{other stuff} \]
\[ \frac{\partial C}{\partial t} = -(k_{cr} C - k_{cf} IC) \]

- Operator asks for C, l as input and looks up \( k_{cr}, k_{cf} \) in the database.
- It then computes a reaction rate \( F = (k_{cr} C - k_{cf} IC) \) and adds it with a + sign to the C equation and a - sign to the l equation.
- It then computes \( \partial F/\partial C = k_{cr} - k_{cf} l \) and stores it, and similarly \( \partial F/\partial l = k_{cf} C \)
Example phyterm

\[ f(c_1, c_2, c_3) = k_f c_1 c_3 - k_r c_2 \]

```
integer function odef( path, ireg, nn, dim, ifunc, ideriv
                   + nfun, mfun, nsol, msol,
                   + sol, gsol, f, df, dgf)
   integer path, ireg, dim, nfun, nsol, ifunc, ideriv, nn
   integer mfun(nfun), msol(nsol)
   double sol(nn,nsol), gsol(nn,dim,nsol)
   double f(nn,nfun), df(nn,nfun,nsol)
   double dgf(nn,dim,dim,nfun,nsol)

   name1 = idmat(ireg) // idvar(msol(1)) // idvar(msol(3)) // 'kf'
   name2 = idmat(ireg) // msol(2) // 'kr'
   libeval('library/physics/' // name1, kf)
   libeval('library/physics/' // name2, kr)

   do 300 in=1,nn
      if( ifunc .ne. 0) then
         bfunc =  kf*sol(in,1)*sol(in,3) - kr*sol(in,2)
         f(in,1) = -bfunc
         f(in,2) =  bfunc
         f(in,3) = -bfunc
      endif
      if( ideriv .ne. 0) then
         dbfdb=  kf*sol(in,3)
         dbfdi=  kf*sol(in,1)
         dbfdbi= -kr
         df(in,1,1)= -dbfdb
         df(in,1,2)= -dbfdi
         df(in,1,3)= -dbfdi
         df(in,2,1)=  dbfdb
         df(in,2,2)=  dbfdi
         df(in,2,3)=  dbfdi
         df(in,3,1)= -dbfdb
         df(in,3,2)= -dbfdi
         df(in,3,3)= -dbfdi
      endif
   300  continue
   odef=0
   return
end
```

• Can be in F77, C, C++, ...
Phyterm arguments

- Every phyterm gets a standard list of arguments `arglist` and their descriptions `argdescrip`.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *path</td>
<td>whether to compute the flux (FT_RUN) or do set-up or clean-up</td>
</tr>
<tr>
<td>int *imtx</td>
<td>whether to compute the flux or its derivatives: 1=flux 10=derivative 11=both</td>
</tr>
<tr>
<td>int *ireg</td>
<td>index of region</td>
</tr>
<tr>
<td>int *nn</td>
<td>number of nodes to work on</td>
</tr>
<tr>
<td>int *dim</td>
<td>space dimension of operator</td>
</tr>
<tr>
<td>int *nsol</td>
<td>number of output variables</td>
</tr>
<tr>
<td>int *msol</td>
<td>indices of each output variables in the global list</td>
</tr>
<tr>
<td>int *ndep</td>
<td>number of input variables</td>
</tr>
<tr>
<td>int *mdep</td>
<td>indices of each input variable in the global list</td>
</tr>
<tr>
<td>real *coord</td>
<td>coordinates of points - for models which have an explicit spatial dependence</td>
</tr>
<tr>
<td>real *sol</td>
<td>the input variables, ordered with node index fast and variable index slow</td>
</tr>
<tr>
<td>real *gradsol</td>
<td>gradients of the inputs, for computing fluxes (nsol,dim,nn)</td>
</tr>
<tr>
<td>real *f</td>
<td>the output fluxes (nsol,dim,nn)</td>
</tr>
<tr>
<td>real *df</td>
<td>derivative of output fluxes with respect to inputs (ndep,nsol,dim,nn)</td>
</tr>
<tr>
<td>real *dgf</td>
<td>derivative of output fluxes with respect to input gradients (ndep,nsol,dim,nn)</td>
</tr>
</tbody>
</table>
Database Features

• Uniform access to
  - coefficients
  - tables
  - user input
  - control options

• Easy to define new parameters in a module without reference to other sections of simulator

• Inheritance allows easy extensions

• Database can be centralized in one file or distributed over several files as desired

• In-memory modified database can be dumped and used for subsequent simulations
Inheritance & Shadowing

Before execution

library/physics {
    silicon {
        boron {
            Dix = 4.2
        }
        poly {
            SeeAlso "../silicon"
        }
    }
}

• find_property("library/physics/poly/boron/Dix") returns a property with value 4.2

• find_list("library/physics/poly/boron/Dix") returns a modified poly list on which a new Dix property can be defined, while still inheriting other properties from silicon

After execution

library/physics {
    silicon {
        boron {
            Dix = 4.2
        }
        poly {
            SeeAlso "../silicon"
            boron {
                SeeAlso "../silicon/boron"
            }
        }
    }
}
database = {
    userInput {
        substrate {
            thick = 2.0
            orientation = 0.1
        }
        graph {
            ...
        }
    }
    options {
        timestep = 1
        movie = "boron,silicon"
    }
    library {
        physics {
            silicon {
                ni = 1.0e10
            }
            boron {
                Dix = 42.0
            }
        }
        math {
            ...
        }
        cards {
            ...
        }
        cards.defaults {
            ...
        }
    }
}
Database subroutines

Library access:

`findDB` (pathname, noComplaint)
  look up pathname and return property

  e.g.  `findDB("library/physics/silicon/boron/Dix", 1)`
  `findDB("solve/temperature", 1)`
  `findDB("options/movie", 0)`

`matco` (coeff, variable n., region n., dom)
  build string `library/physics/region/variable/coeff`
  and call `findDB` to return property

Building/accessing properties lists:

`get_property` (name, list)
`get_local_property` (name, list)
`get_next_local_property` (name, list, start)
`put_local_property` (name, list)
`put_local_list` (name, list)

`count_properties` (list)
`get_property_name` (list, index)
`get_property_by_index` (list, index)

`delete_local_property` (name, list)
`delete_plist` (name, list)

See `solvecmd()` for example of use.
Grid datastructure design

- Core representation is minimal and unstructured
  - Facilitates adding and removing elements, nodes
  - Each node, element, list of vertices in an element, etc, is malloc’ed separately, in any order

- Automatic routines to build inferred information from core representation
  - Internal neighbors
  - List of unique edges
  - Interface structures
  - Reorganization into similarity groups for vector access

Within a color, all elements have same region and type
All indices taken from single malloc block
Similar for coordinates, neighbors, etc
Grid datastructures (core)

- Finite element grid representation, with special handling of material boundaries

**Domain**

<table>
<thead>
<tr>
<th>List of elements</th>
<th>region number, node n’s, nbr n’s, edge n’s,</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of nodes</td>
<td>point number</td>
</tr>
<tr>
<td>List of points</td>
<td>coordinates</td>
</tr>
<tr>
<td>List of edges</td>
<td>node n’s [2]</td>
</tr>
<tr>
<td>List of fields</td>
<td>name {boron, potential, \ldots}</td>
</tr>
<tr>
<td></td>
<td>type {node, edge, element}</td>
</tr>
<tr>
<td></td>
<td>dimension {1, 2, 3}</td>
</tr>
<tr>
<td>List of region and interface names</td>
<td>{silicon, periodic, \ldots}</td>
</tr>
</tbody>
</table>

- Boundary conditions represented by external element neighbors being negative, indices into list of interface names
Grid datastructures 
(derived)

- Interface structures can be generated on demand

Domain

... List of interfaces
  Reg n's [2]
  Element count nel
  Node count nno
  List of elements [nel*2] and faces[nel*2]
  List of nodes [nno*2]

- Each "element" of interface identifies which bulk element and which face of that element are on the interface

- Each "node" of interface lists the two corresponding nodes

build_itf(dom)

Domain + 1 red interface + 1 green interface + 1 blue interface + 3 black interfaces
Grid subroutines

Alloc_domain()
Alloc_domain_region( dom, name)
Alloc_domain_surface (dom, name)
ixmaterial( dom, name) - index of material in region (first if multiple)
ixsurface( dom, name) - index of surface in region (first if multiple)

alloc_node (dom, n)
alloc_point (dom, n)
alloc_edge (dom, n)
alloc_element (dom, n, type, ireg)

alloc_field( dom, type, name, dim, initvalue)
free_field( dom, index)
ixfield( dom, name)

build_nds( dom) - if given elements in terms of points, convert to node description
(convenient for reading from conventional finite element grid)
build_edge( dom) - given elements and nodes, build list of unique edges
build_nbrs( dom) - given elements, build internal neighbors
build_itf( dom) - build list of unique interfaces
lock_domain( dom, andColor) - reorder for vector access
unlock_domain( dom)

domain_sanity_check( dom)
solvecmd
  make_diff_list
  steptime
    solcontrol
    diff_setup
      lock_domain
      build_itf
      shadowDom
      variable-specific initialization
        usually matrix_init()
      boxGeom
      boundary_cond
    trbdf2_step
    OuterGS
      setFlux(FT_DT)
    innerNewton
      assemble
      mtxscl
      c2b1s
      update_vars
    OuterGS
    OuterGS
    milne
assemble

assemble_alloc
assemble_space
assemble_eliminate
assemble_elementals

  call divtrm to discretize laplacian
  interpolate to quad point
  call a phyterm
  store to window

  each element
  color

  call addstf to transfer window to matrix

assemble_nodals

  pdediag for each region

  copy variables to contiguous storage
  call a phyterm
  store to rhs, matrix

assemble_box

  each edge
  color

  interpolate to midpoints
  call a phyterm
  store to rhs, matrix

assemble_itf_nodals

  similar

assemble_time
condenseItf
PDE description - internal

- Output of **solcontrol**

Database description  

<table>
<thead>
<tr>
<th>globsolstruc: for each field of grid</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sol_type</strong> {general, pdesteady, pdetransient, formula}</td>
</tr>
<tr>
<td>setup routine (e.g. special Tdep initialization for interstitials)</td>
</tr>
<tr>
<td>refresh routine</td>
</tr>
<tr>
<td>teardown routine</td>
</tr>
<tr>
<td>block of pde’s u.pb</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nsol number of solution variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>sol list of solution variables</td>
</tr>
<tr>
<td>nterm number of terms</td>
</tr>
<tr>
<td>term[0]</td>
</tr>
</tbody>
</table>

| geoterm |
| phyterm |
| number and list of input variables |
| number and list of output equations |
| output-input coupling flags |
| between[ni] - which interfaces (if BC term) |

| term[1]... |
| mtxtype - block sparsity array |
| linear solution options |
Timestep layout

- Grid structural changes happen between timesteps
- Grid motion happens during timestep
- "Timestep" is composite TRBDF2 step

**Diagram:**
- TRBDF2 step diagram with grid structural changes and grid motion.
- "Rezone" indicates a change in grid structure.
Sparse Matrix Structure  
(Bank/Smith)

- Represent graph structure only once
- Represent PDE structure by hierarchy

\[
\begin{align*}
  \frac{\partial B}{\partial t} &= D_B \nabla^2 B + f(B, \psi) \\
  \frac{\partial C}{\partial t} &= D_C \nabla^2 C + g(C, \psi) \\
  \epsilon \nabla^2 \psi &= h(B, C)
\end{align*}
\]

Equations

\[\text{Grid}\]

\[\begin{array}{cccccccc}
  0 & 4 & 8 & 12 \\
  1 & 5 & 9 &   \\
  2 & 6 & 10 &   \\
  3 & 7 & 11 &   \\
\end{array}\]

\[\text{ia} \quad 17 \ 20 \ 23 \quad 14 \ 5 \ 2 \ 5 \ 6 \ 3 \ 6 \ 7\ldots\]

\[0 \quad 16 \ 17\]

pointers connections

real storage for one block "aa"

- diagonal
- \(z\)
- upper triangle
- lower triangle

\[\text{loca}[i+n\text{b}^j] \text{ is location of matrix for ith equation, jth variable}\]

All storage taken from single matrix array \(a\)

e.g. diagonal \#10 of C equation wrt B variable is \(a[\text{loca}[1]+10]\)

Right hand side \(\text{rhs}\) stored \(B_0, C_0, \psi_0, B_1, C_1, \psi_1,\ldots\)
PROPHET Genealogy

Old Testament

Remove specific process modules

New Testament

New database
New gridstructure
New device modules

Lucent process modules

Process simulator

PDE solver

2Q96  3-4Q96
↓   ↓
1-2Q97
↓
3Q97
↓
4Q97
↓

PDE solver
### Directory structure

**src/**
- **Dbase/** database implementation
- **Grid/** grid implementation
- **Guide/** input parser
- **Main/**
- **Misc/** common macros
- **Mod/** modules
  - **Bias/** device simulation
  - **Boundary/** define BC’s
  - **Dbase/** modify database contents
  - **Dump/** raw output
  - **Field/** define fields over grid
  - **Graph/** 1D/2D graphics
  - **Grid/** define working grid
  - **Solve/** call solver
  - **xgraph/** X-based graphics tool
- **PDE/**
  - **Casmbly/** assembly control
  - **Fasmbly/** discretization
  - **Fluxes/** phyterms
  - **Vexpr/** expression parser
- **lib/**
  - **dbase.prophet** text of database
- **arch/** .a and .exe - per architecture

---

*Colored = soon to be replaced by TCL/TK*