GLAF: A Visual Programming and Auto-Tuning Framework for Parallel Computing

Student: Konstantinos Krommydas

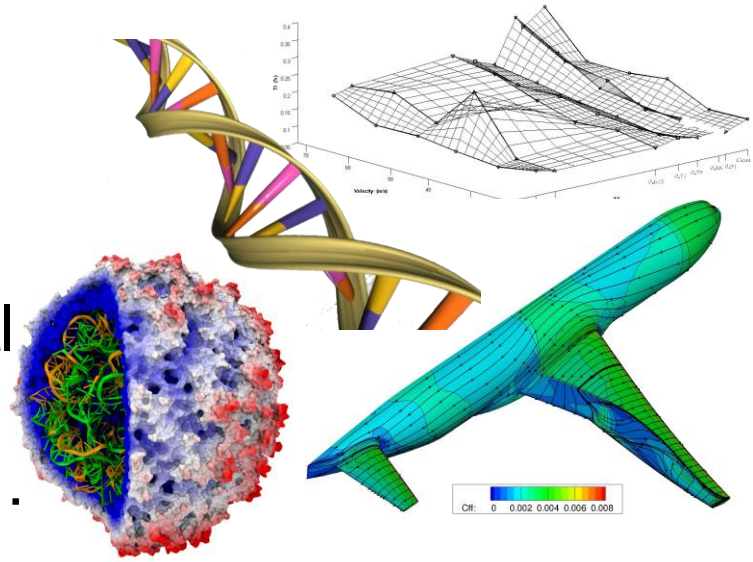
Collaborator: Dr. Ruchira Sasanka (Intel)

Advisor: Dr. Wu-chun Feng

VirginiaTech
Invent the Future

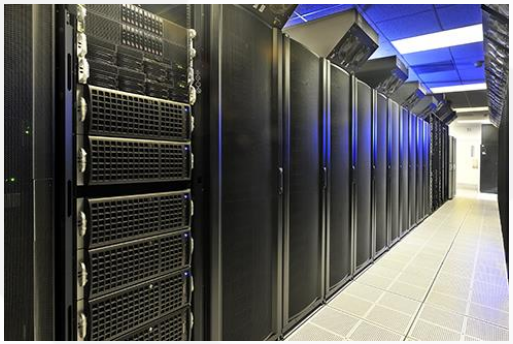
SyNeRG
synergy.cs.vt.edu

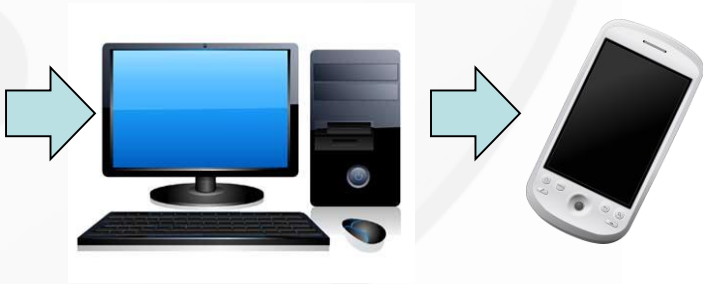
Motivation



High-performance computing is crucial in a broad range of scientific domains:

- Engineering, math, physics, biology, …





Parallel programming revolution has made high-performance computing **accessible** to the broader masses

Challenges

Many *different* computing platforms

Many different programming languages

Many different architectures

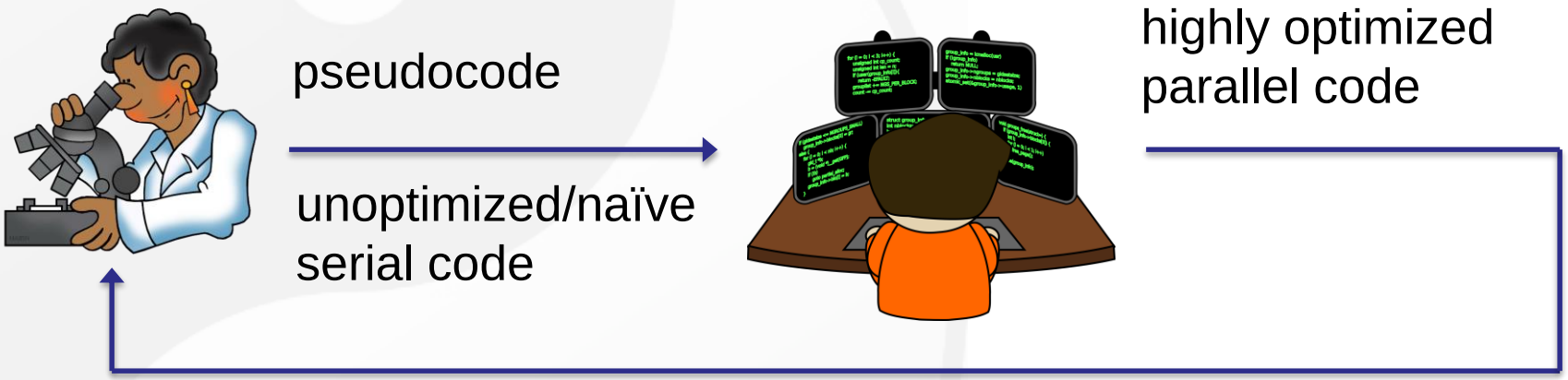
Many different optimization strategies

Domain experts **should not** (need to) know all these details

- but rather focus on **their** science

Challenges

Domain experts need to *collaborate* with **computer scientists**



- Communication overhead & errors
- Need to exchange domain-specific/programming knowledge

Innovation slow-down

Limited access to parallel computing

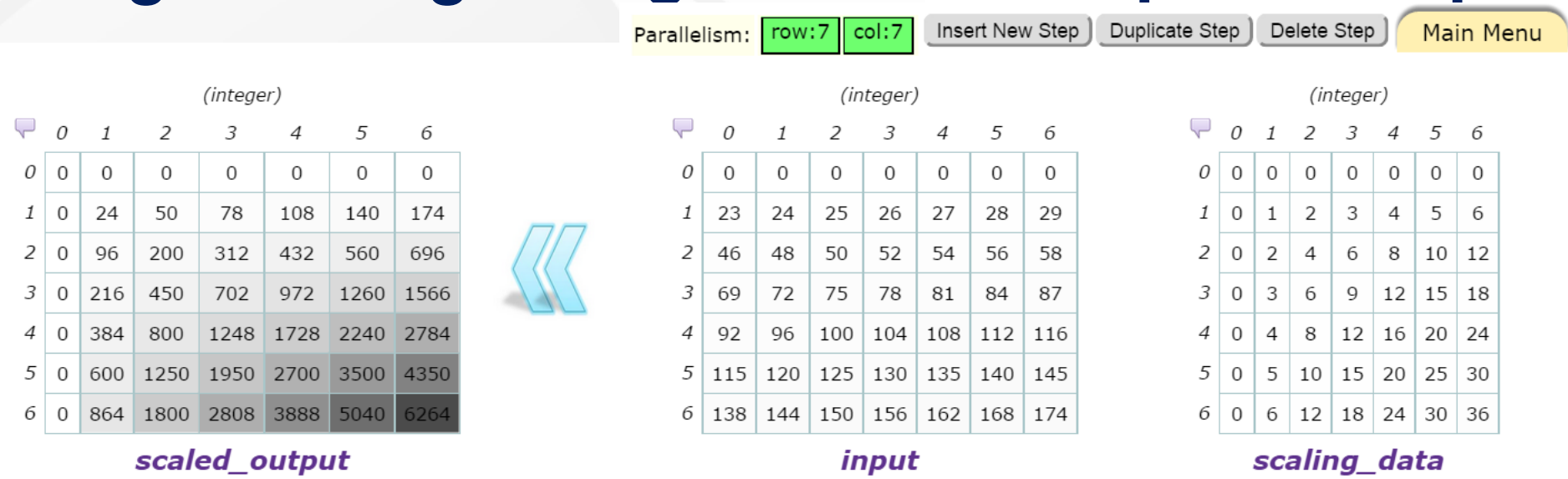
Contributions

- Realize a programming abstraction & development framework for domain experts to provide a balance between performance and programmability
  - i.e., obtain fast performance from algorithms that have been programmed easily

Desired features

*GLAF
intuitive, familiar, minimalistic syntax
data-visual and interactive
auto-parallelizable, optimizable, tunable
able to integrate with existing legacy code

Programming Using GLAF: a Simple Example

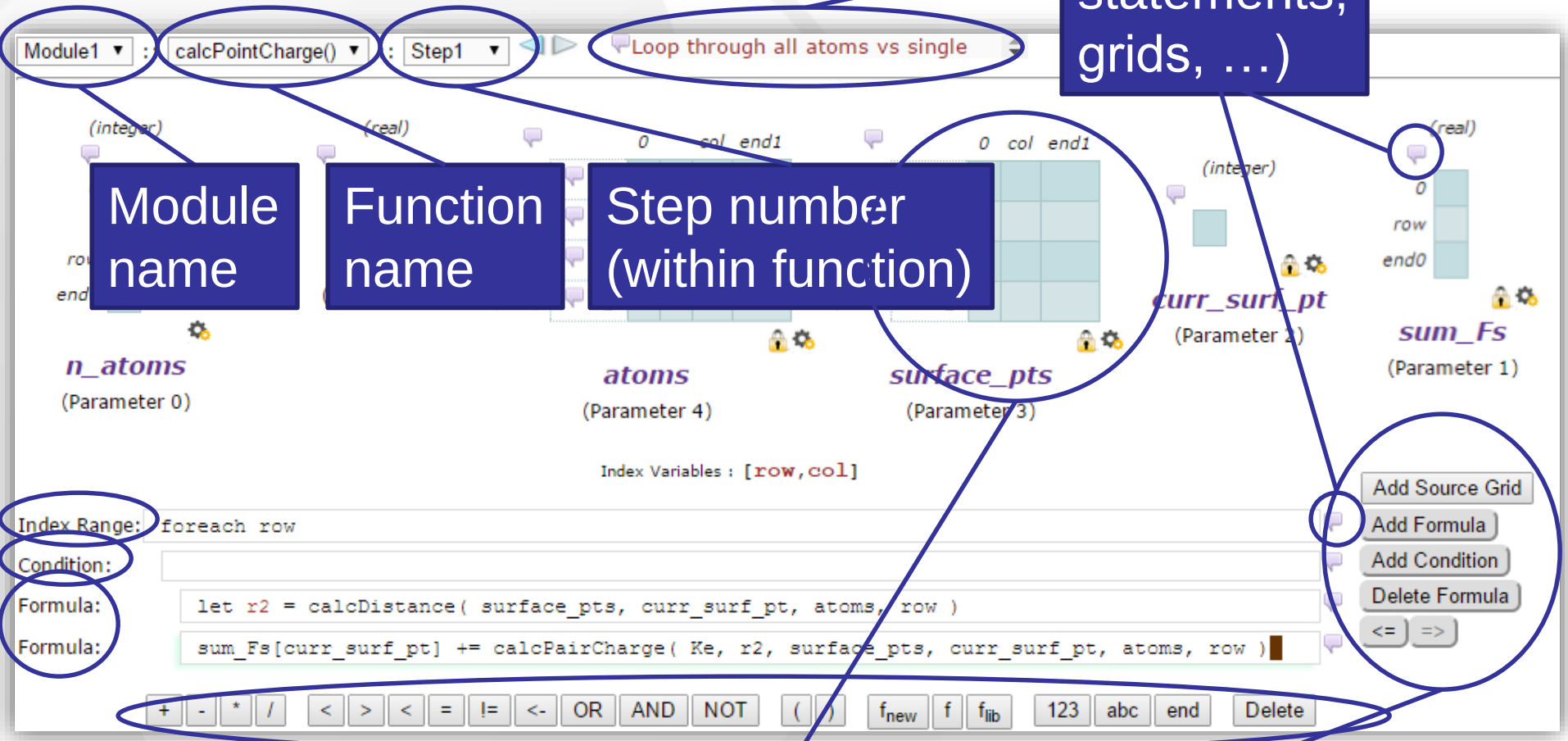


	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	24	50	78	108	140	174
2	0	96	200	312	432	560	696
3	0	216	450	702	972	1260	1566
4	0	384	800	1248	1728	2240	2784
5	0	600	1250	1950	2700	3500	4350
6	0	864	1800	2808	3888	5040	6264

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	23	24	25	26	27	28	29
2	46	48	50	52	54	56	58
3	69	72	75	78	81	84	87
4	92	96	100	104	108	112	116
5	115	120	125	130	135	140	145
6	138	144	150	156	162	168	174

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	8	10	12
3	0	3	6	9	12	15	18
4	0	4	8	12	16	20	24
5	0	5	10	15	20	25	30
6	0	6	12	18	24	30	36

Graphical User Interface



Module name

Function name

Step number (within function)

Comments (in step, statements, grids, …)

Click-based interface

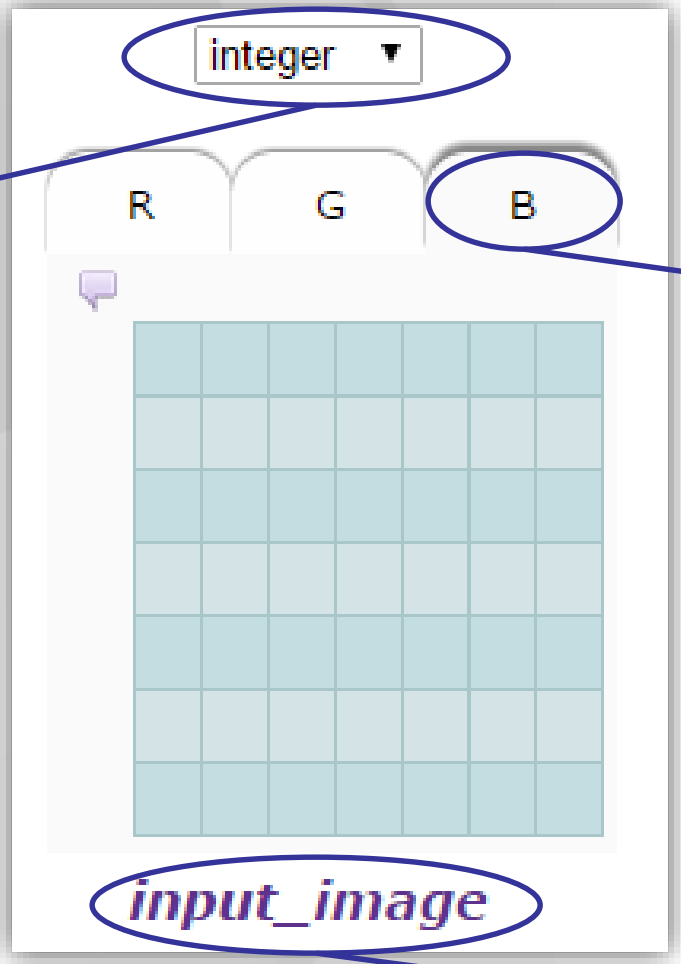
Grid-Based Data Structures

- GLAF variables are based on the concept of **grids**:
  - familiar abstraction (e.g., images, matrices, spreadsheets)
  - regular format that facilitates code generation, optimizations, and parallelism detection

- **Grid-based programming** puts the focus on the **relation**, rather than the implementation details

- **Example:**
  - *Scalar variable*: 0D grid
  - *1D array*: one-dimensional grid

Grid-Based Data Structures



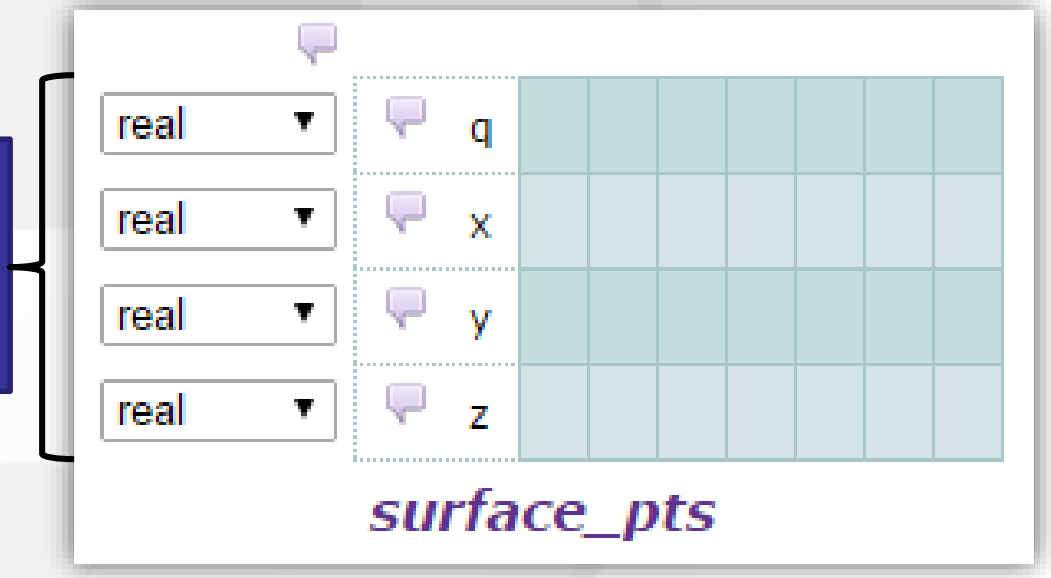
Data type

Dimension title

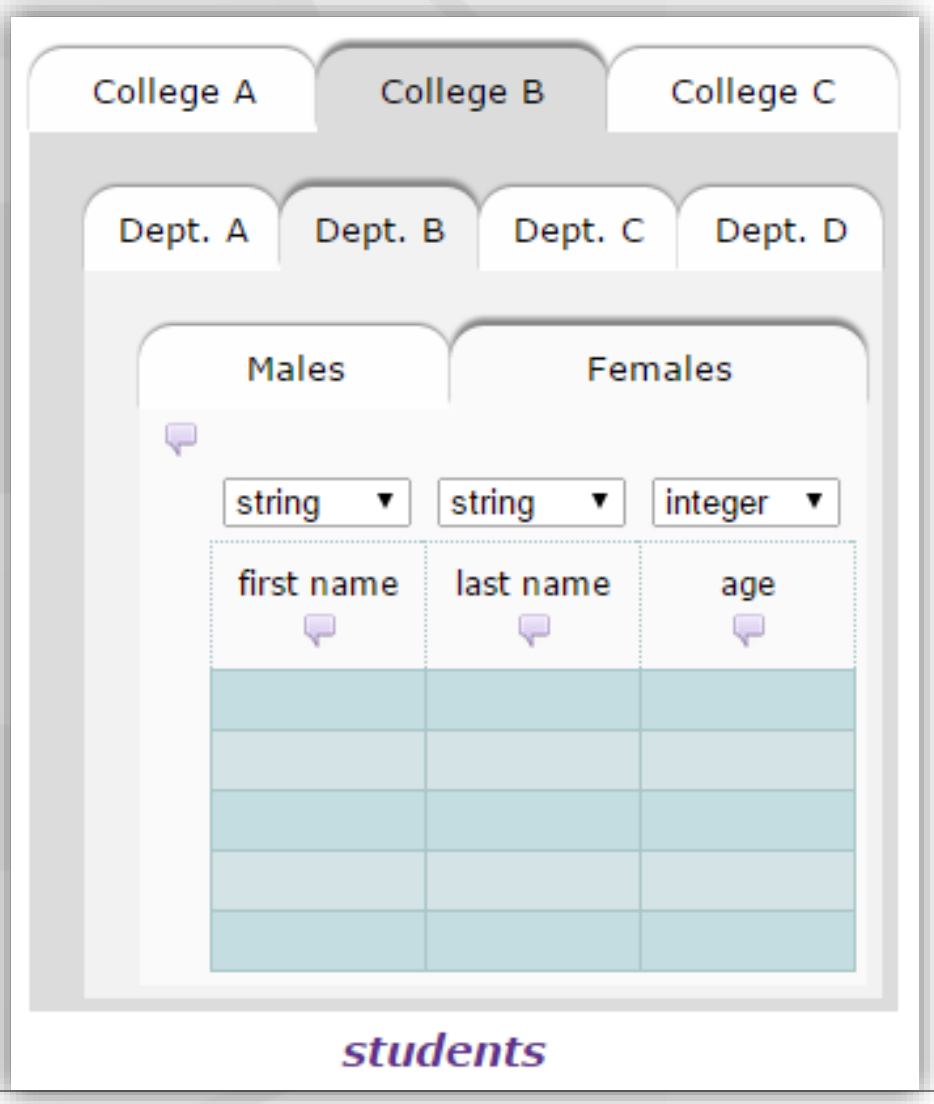
Grid name

Grid-Based Data Structures

Different data types across a dimension



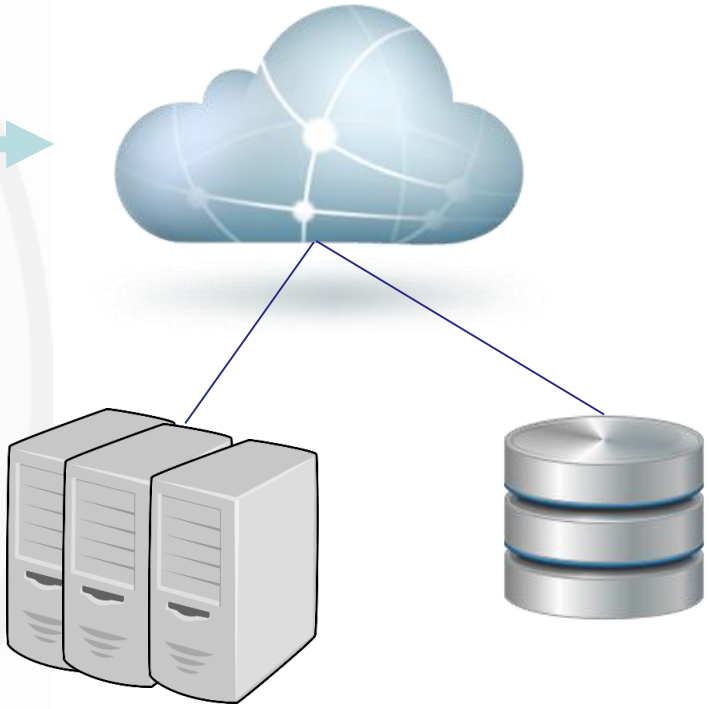
Grid-Based Data Structures



GLAF Infrastructure

Browser

Web/Cloud



- GLAF Programming GUI
- Data visualization
- Fortran/C/JS/OpenCL code generation
- Auto-parallelization
- Compilation/auto-tuning script generation

- Compilation
- Auto-tuning
- Execution

- Data storage

Code Generation

```
int ft_calcPointCharge(int *ft_n_atoms, float *ft_sum_Fs, int ft_curr_surf_pt, struct TYP_surface_pts typvar_surface_pts, struct
TYP_surface_pts typvar_atoms, float ft_Ke) {
	int fun_curr_surf_pt;
	float fun_Ke;
	int ft_ReturnValue;
	int ft_row;
	int ft_end0;
	int atoms_q=0, atoms_x=1, atoms_y=2, atoms_z=3;
	int surface_pts_q=0, surface_pts_x=1, surface_pts_y=2, surface_pts_z=3;
	float ft_r2;
	fun_curr_surf_pt = ft_curr_surf_pt;
	fun_Ke = ft_Ke;
	// Loop through all atoms vs single surf pt
	ft_end0 = 4-1;
	float tmp_sum_Fs=ft_sum_Fs[fun_curr_surf_pt];
	#pragma omp parallel for collapse(1) private(ft_r2) reduction(+: tmp_sum_Fs)
	for (ft_row = 0; ft_row <= ft_end0; ft_row += 1) {
		// Calculating the distance between a surface point and each atom
		ft_r2 = ft_calcDistance(typvar_surface_pts,fun_curr_surf_pt,typvar_atoms,ft_row);
		// Add current pairs charge to total
		tmp_sum_Fs = tmp_sum_Fs + ft_calcPairCharge(fun_Ke,ft_r2,typvar_surface_pts,fun_curr_surf_pt,typvar_atoms,ft_row);
	}

	ft_sum_Fs[fun_curr_surf_pt]=tmp_sum_Fs;
}
```

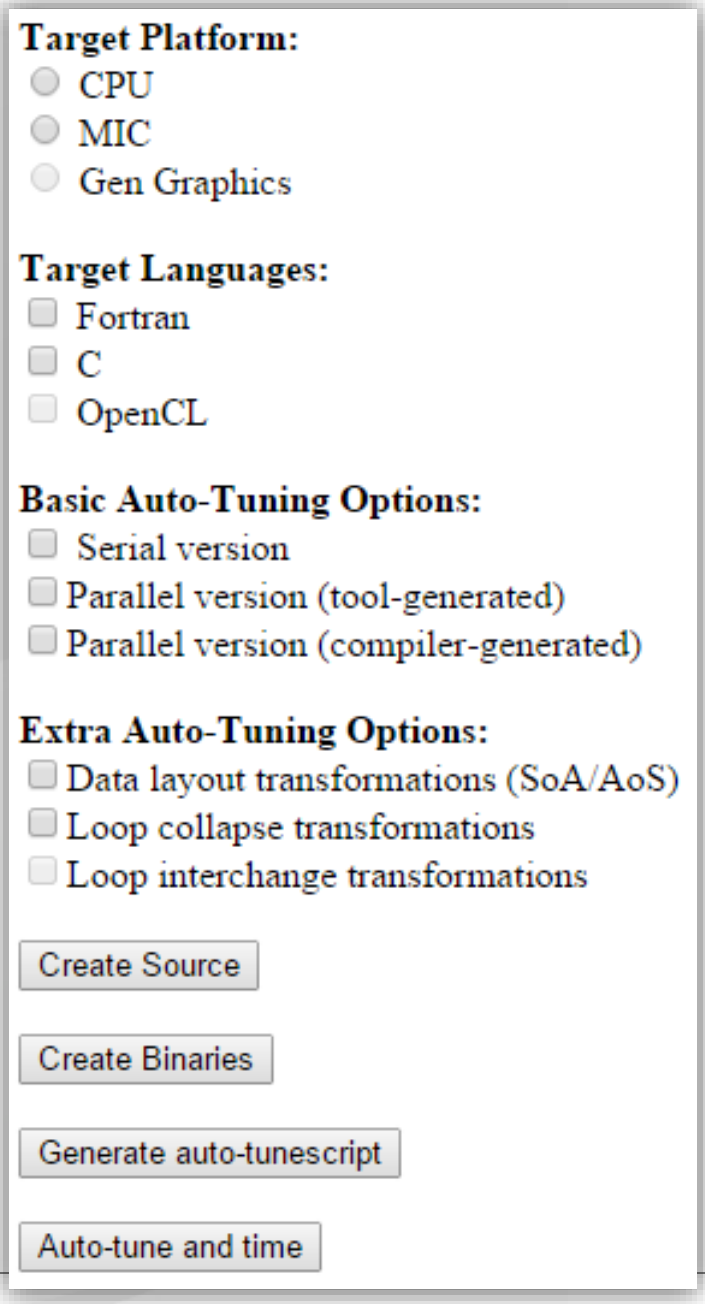
Code Generation

```
INTEGER FUNCTION ft_calcPointCharge(ft_n_atoms,ft_sum_Fs,ft_curr_surf_pt,typvar_surface_pts,typvar_atoms,ft_Ke)
       INTEGER, DIMENSION(:) :: ft_n_atoms
       REAL, DIMENSION(:) :: ft_sum_Fs
       INTEGER :: ft_curr_surf_pt
       TYPE (TYP_surface_pts) typvar_surface_pts
       TYPE (TYP_atoms) typvar_atoms
       REAL :: ft_Ke
       INTEGER :: fun_curr_surf_pt
       REAL :: fun_Ke
       INTEGER ::ft_ReturnValue
       INTEGER :: ft_row
       INTEGER :: ft_end0
       INTEGER :: atoms_q=0, atoms_x=1, atoms_y=2, atoms_z=3
       INTEGER :: surface_pts_q=0, surface_pts_x=1, surface_pts_y=2, surface_pts_z=3
       REAL ft_r2
       fun_curr_surf_pt = ft_curr_surf_pt
       fun_Ke = ft_Ke
       !Loop through all atoms vs single surf pt
       ft_end0 = 4-1
       !$OMP PARALLEL DO COLLAPSE(1) &
       !$OMP REDUCTION(+: ft_sum_Fs) &
       !$OMP PRIVATE(ft_r2)
       DO ft_row = 0, ft_end0, 1
              ! Calculating the distance between a surface point and each atom
              ft_r2 = ft_calcDistance(typvar_surface_pts,fun_curr_surf_pt,typvar_atoms,ft_row)
              ! Add current pairs charge to total
              ft_sum_Fs(fun_curr_surf_pt + 1) = ft_sum_Fs(fun_curr_surf_pt + 1) + ft_calcPairCharge(fun_Ke,ft_r2,typvar_surface_pts
,fun_curr_surf_pt,typvar_atoms,ft_row)
       END DO
       !$OMP END PARALLEL DO
       ft_calcPointCharge = ft_ReturnValue
       RETURN
END FUNCTION
```

Auto-Tuning

Selects the ***languages*** in which to auto-generate code

Selects optimizations for ***each*** combination of language and code “starting point”



Generates ***platform-specific*** binaries & optimizations

Selects one or more code “starting points” for ***each*** language

Auto-Tuning

Target Platform:
- ○ CPU
- ○ MIC
- ○ Gen Graphics

Target Languages:
- ☐ Fortran
- ☐ C
- ☐ OpenCL

Basic Auto-Tuning Options:
- ☐ Serial version
- ☐ Parallel version (tool-generated)
- ☐ Parallel version (compiler-generated)

Extra Auto-Tuning Options:
- ☐ Data layout transformations (SoA/AoS)
- ☐ Loop collapse transformations
- ☐ Loop interchange transformations

Create Source

Create Binaries

Generate auto-tunescript

Auto-tune and time

```
struct TYP_surface_pts {
  float *dim0;
  float *dim1;
  float *dim2;
  float *dim3;
};

// Surface points outside the biomolecule
struct TYP_surface_pts typvar_surface_pts;

// Allocation of each array within the struct (SoA)
typvar_surface_pts.dim0=(float *)malloc(sizeof(float )*3);

// Value assignment
typvar_surface_pts.dim0[ft_col] = ft_col * 3;
```

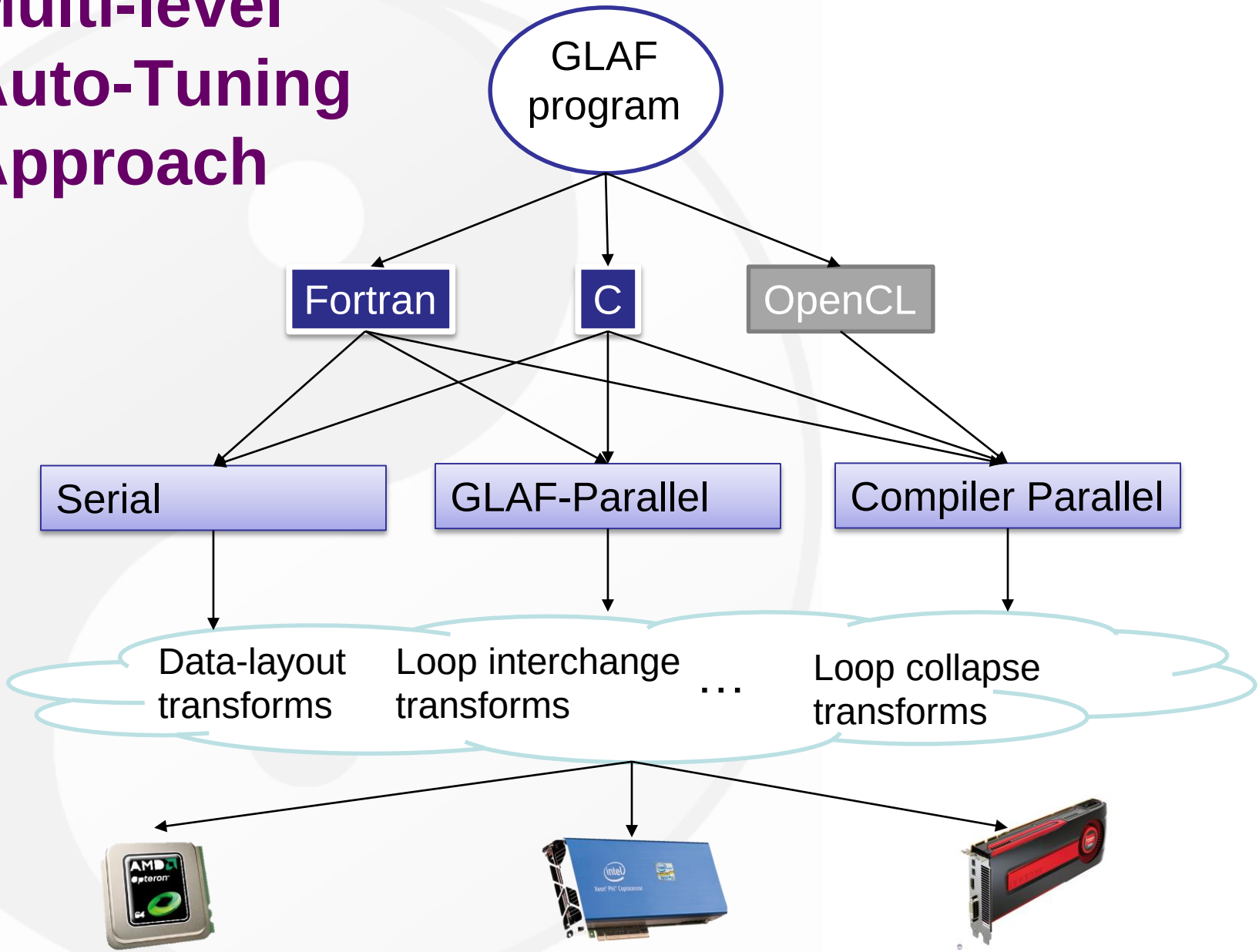
```
struct TYP_surface_pts {
  float dim0;
  float dim1;
  float dim2;
  float dim3;
};

// Surface points outside the biomolecule
struct TYP_surface_pts *typvar_surface_pts;

// Initialization of array of structures (AoS)
typvar_surface_pts = (struct TYP_surface_pts *)malloc(
                     sizeof(struct TYP_surface_pts)*3);

// Value assignment
typvar_surface_pts[ft_col].dim0 = ft_col * 3;
```

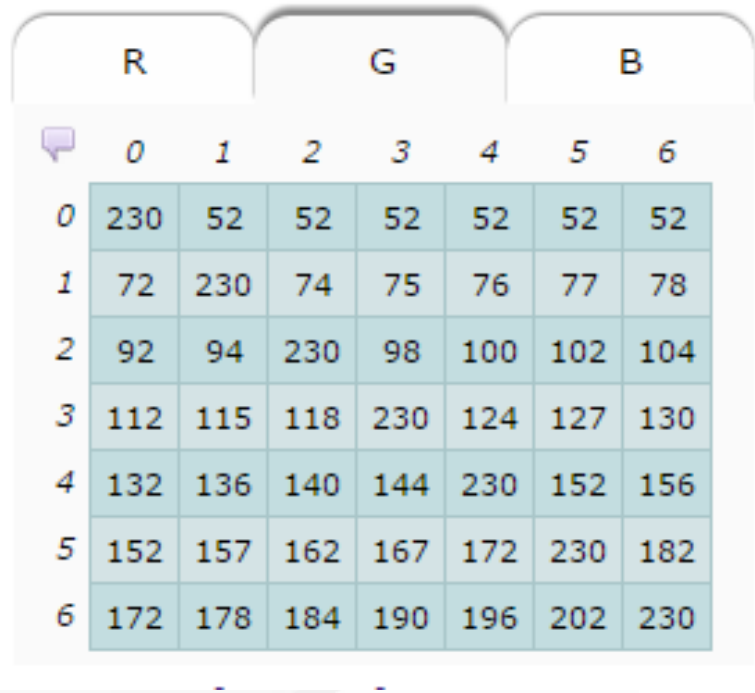
Multi-level Auto-Tuning Approach



Visualization

- Data visualization facilitates:
  - understanding the algorithm being developed
  - revealing bugs at an early stage

"Show Data"

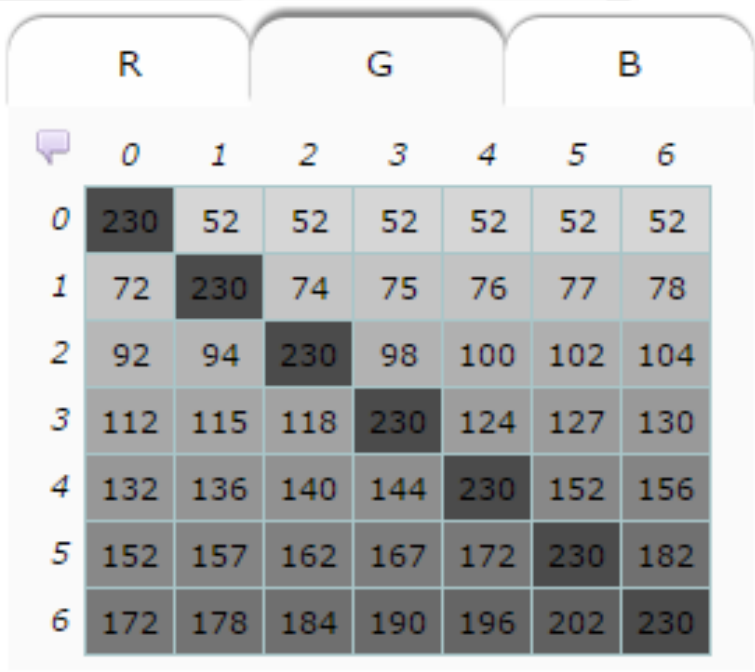


	R			G			B
	0	1	2	3	4	5	6
0	230	52	52	52	52	52	52
1	72	230	74	75	76	77	78
2	92	94	230	98	100	102	104
3	112	115	118	230	124	127	130
4	132	136	140	144	230	152	156
5	152	157	162	167	172	230	182
6	172	178	184	190	196	202	230

Visualization

- Data visualization facilitates:
  - understanding the algorithm being developed
  - revealing bugs at an early stage

“Colorize”

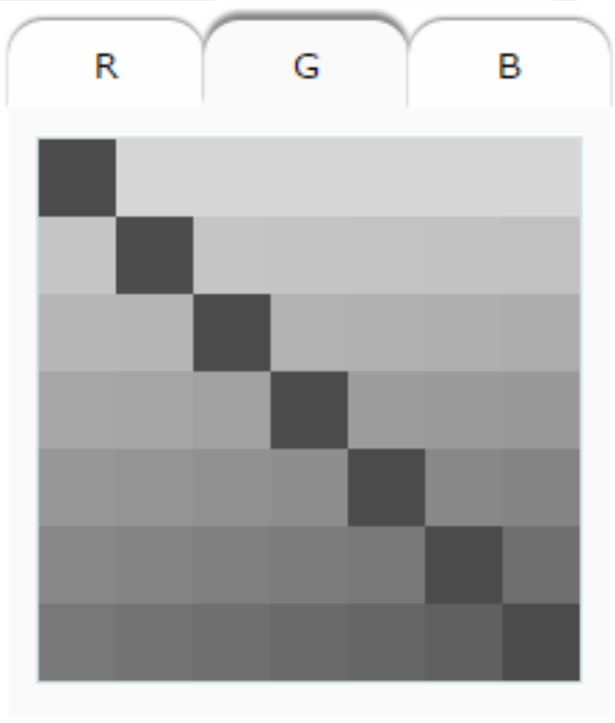


	0	1	2	3	4	5	6
0	230	52	52	52	52	52	52
1	72	230	74	75	76	77	78
2	92	94	230	98	100	102	104
3	112	115	118	230	124	127	130
4	132	136	140	144	230	152	156
5	152	157	162	167	172	230	182
6	172	178	184	190	196	202	230

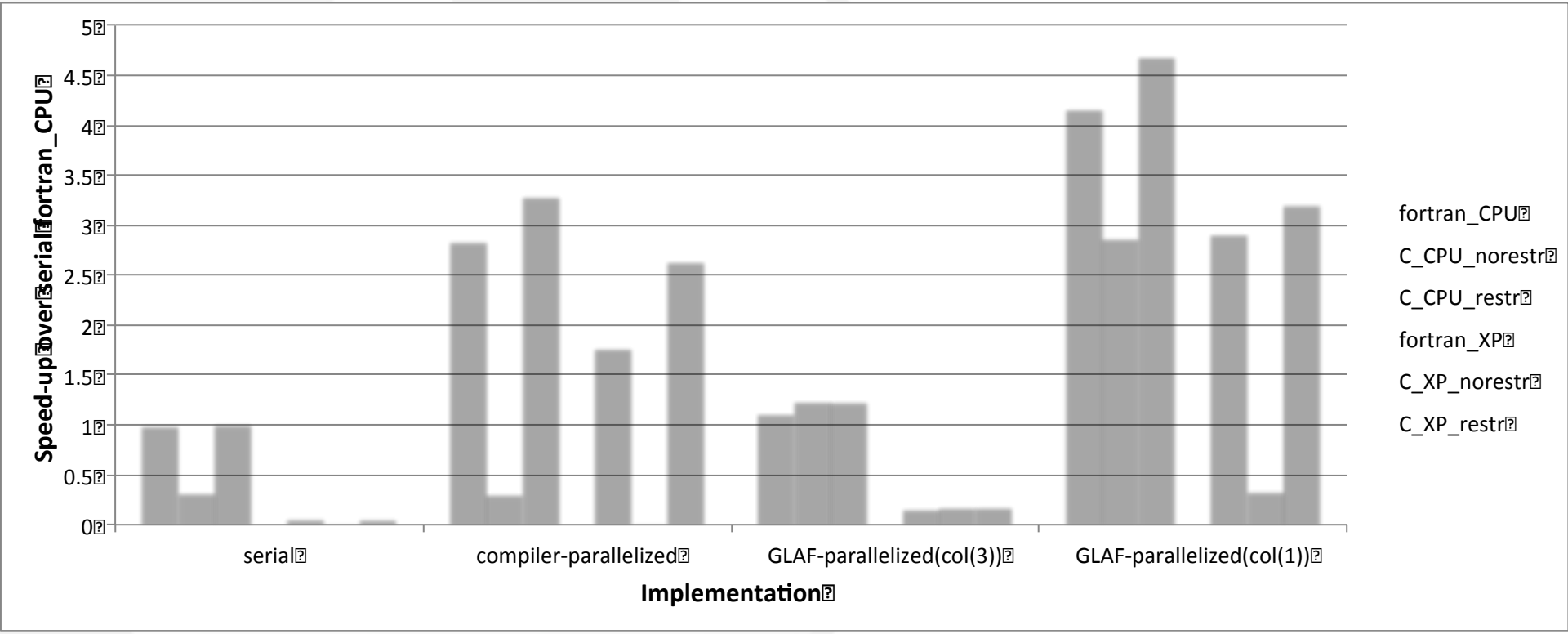
Visualization

- Data visualization facilitates:
  - understanding the algorithm being developed
  - revealing bugs at an early stage

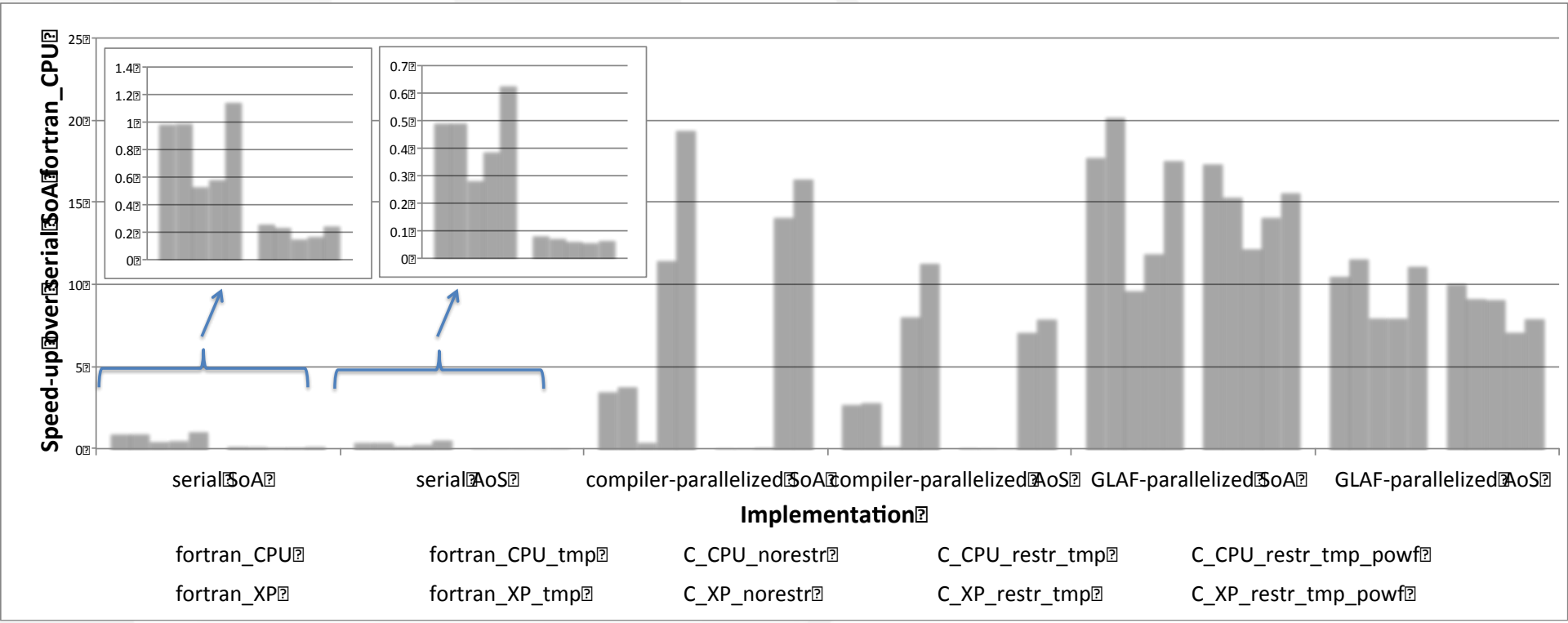
"Image Map"



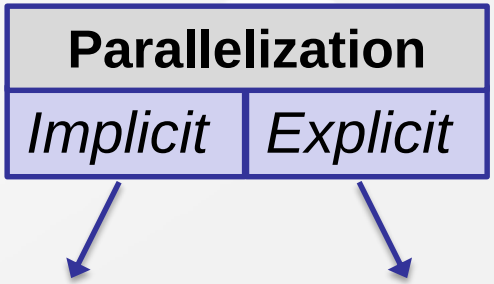
Results: 3D finite difference algorithm



Results: N-body algorithm



Related Work



Compilers:
gcc, Intel, Cray, PGI, …

Extensions/ libraries:
Pthreads, OpenMP, OpenACC, Chapel …

Problem Solving Environments	Domain-Specific Languages

Domain-specific:
computational biology, physics,
dense linear algebra,
Fast-fourier transforms,
…

+ Includes loop-level parallelism & data-level parallelism (vectorization), and potential optimizations
- Requires certain programming knowledge
- (Often) conservative nature

+ High-performance auto-tuning
- Restrictive in nature
- Restrictive in terms of target language and/or platform

Future Work

- Improve tool's **robustness**
- Enabling **more languages/extensions**:
  - OpenCL, OpenACC
  - Support of distributed programming (MPI)
- Dynamic **feedback/advice** on parallelism issues
- **Extend** auto-tuning, auto-parallelization/auto-vectorization capabilities
- Implement **more dwarfs** and provide back-end support for common programming **pitfalls** in code generated for supported languages

Conclusion

GLAF targets domain experts and provides a fine balance between **performance** and **programmability**

- auto-parallelization, optimization and auto-tuning
- helps avoid common programming pitfalls

In summary:		**Analogous to:**
GLAF allows systematic generation of **multiple starting points** for different languages/platforms/optimizations	⟷	Different seeds in a state-space search algorithm
Leads to **overall better** performance	⟷	Global vs. local minimum