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

Student: Konstantinos Krommydas
Collaborator: Dr. Ruchira Sasanka (Intel)
Advisor: Dr. Wu-chun Feng
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

*GLAF

Desired features

<table>
<thead>
<tr>
<th>*GLAF</th>
<th>intuitive, familiar, minimalistic syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>data-visual and interactive</td>
</tr>
<tr>
<td></td>
<td>auto-parallelizable, optimizable, tunable</td>
</tr>
<tr>
<td></td>
<td>able to integrate with existing legacy code</td>
</tr>
</tbody>
</table>
Programming Using GLAF: a Simple Example

Index Variables: [row, col]

Index Range: foreach row col

Condition: 

Formula: 

Graphical User Interface

Comments (in step, statements, grids, …)

Module name
Function name
Step number (within function)

Module name: calcPointCharge
Function name: Step 1
Step number (within function): Loop through all atoms vs single

Index Range: foreach row
Condition: let r2 = calcDistance(surface_pts, curr_surf_pt, atoms, row)
Formula: sum_Fs[curr_surf_pt] += calcPairCharge(Ke, r2, surface_pts, curr_surf_pt, atoms, row)

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

input_image
Grid-Based Data Structures

Different data types across a dimension
Grid-Based Data Structures
GLAF Infrastructure

Browser

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

Web/Cloud

- Compilation
- Auto-tuning
- Execution
- Data storage
```c
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;
}
```
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
    !$OMP PARALLEL DO COLLAPSE(1) &
    !$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
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

```c
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

GLAF program

- Fortran
- C
- OpenCL

Serial
GLAF-Parallel
Compiler Parallel

- Data-layout transforms
- Loop interchange transforms
- ... Loop collapse transforms
Visualization

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

“Show Data”
Visualization

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

“Colorize”
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

<table>
<thead>
<tr>
<th>Implementation</th>
<th>serial SoA</th>
<th>serial AoS</th>
<th>compiler-parallelized SoA</th>
<th>compiler-parallelized AoS</th>
<th>GLAF-parallelized SoA</th>
<th>GLAF-parallelized AoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>fortran_CPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fortran_XP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fortran_CPU_tmp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fortran_XP_tmp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_CPU_norestr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_XP_norestr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_CPU_restr_tmp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_XP_restr_tmp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_CPU_restr_tmp_powf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_XP_restr_tmp_powf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Related Work

Parallelization

| Implicit | Explicit |

Compilers: gcc, Intel, Cray, PGI, ...

Extensions/libraries:
- Pthreads, OpenMP, OpenACC, Chapel, ...

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

Problem Solving Environments

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

- 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:
GLAF allows systematic generation of multiple starting points for different languages/platforms/optimizations
Leads to overall better performance

Analogous to:
Different seeds in a state-space search algorithm
Global vs. local minimum

“GLAF: A Visual Programming and Auto-Tuning Framework for Parallel Computing”
Krommydas, Sasanka, Feng