Compiler Supported High-level Abstractions for Sparse Disk-resident Datasets

Renato Ferreira
Gagan Agrawal
Joel Saltz
Ohio State University
General Motivation

Computing is playing an increasingly more significant role in a variety of scientific areas.

Traditionally, the focus was on simulating scientific phenomenon or processes:
- Software tools motivated by various computational solvers.

Recently, analysis of data is being considered key to advances in sciences:
- Data from computational simulations
- Digitized images
- Data from sensors
Challenges in Supporting Processing

- Massive amounts of data are becoming common
  - Data from simulations of large grids, parameters studies
  - Sensors collecting high resolution data, over long periods of time
- Datasets can be quite complex
- Applications scientists need high-performance as well as ease of implementing and modifying analysis
Motivating Application: Satellite Data Processing

- Data collected by satellites is a collection of chunks, each of which captures an irregular section of earth captured at time $t$.
- The entire dataset comprises multiples pixels for each point in earth at different times, but not for all times.
- Typical processing is a reduction along the time dimension - hard to write on the raw data format.
Supporting High-level Abstractions

- View the dataset as a dense 3-d array, where many values can be zero
- Simplify the specification of processing on the datasets
- Challenge: how do we achieve efficient processing?
  - Locality in accessing data
  - Avoiding unnecessary computations
Outline

- Compiler front-end
- Execution strategy for irregular/sparse applications
- Supporting compiler analyses
- Performance enhancements
  - Dense applications
  - Code motion for conditionals
- Experimental results
- Conclusion
Programming Interface

- Multi-dimensional collections
  - Domain
  - RectDomain
- Foreach loop
  - Iterates of the elements of a collection
- Reduction interface
  - Defines reduction variables
    - Update within the foreach
    - Associative and commutative operations
    - Only used for self updates
public class Element {
    short bands[5];
    short lat, long;
}

public class SatelliteApp {
    SatelliteData satdata;
    OutputData output;
    public static void main(String[] args) {
        Point[2] q; pixel val;
        RectDomain[3d] AbsDomain = ... 
        foreach (q in AbsDomain)
        if (val = satdata.getData(q)) {
            Point[2] p = (q[1], q[2]);
            output[p].Accumulate(val);
        }
    }
}
Sparse Execution Strategy

- Iterating over AbsDomain
  - Sparse domain
  - Poor locality
- Iterating over input elements
  - Need to map element to loop iteration

Foreach element e
  I = Iters(e)
  Foreach i in I
    If (i in the Input Range)
      Perform computation for e
Computing function `Iters()`

- `Iters` (element -> abstract domain)
  - l-value of element = `<t, i>`
  - r-value of element = `<b1, b2, b3, b4, b5, lat, long>`
  - `Iters(elem = <l-value, r-value>)` → `<t, lat, long>`

- Find the dominating constraints for the return statements within the functions in the low-level data layout (getData)
(Chunk-wise) Dense Strategy

- Exploit the regularity on the dataset
  - Eliminate overhead of sparse strategy
- Simpler, more efficient implementation

Foreach input block

Extract D (descriptor of the data)

\[ I = \text{Iters}(D) \cap \text{Input Range} \]

Foreach i in I

Perform computation for Input[i]
Other Implementation Issues

- Generating code for efficient execution
  - ADR run-time system
- Memory requirements
  - Tiling of the output
- Extract subscript and range functions from user application
  - Program Slicing (ICS 2000)
- Compiler and runtime communication analysis (PACT 2001)
Active Data Repository

- Specialized run-time support for processing disk-based multi-dimensional datasets
  - Push processing into storage manager
  - Asynchronous operations

- Dataset is divided in blocks
  - Distribute across the nodes of a parallel machine
  - Spatial indexing mechanism

- Customizable for a variety of applications
  - Through virtual functions
  - Supplied by the compiler
Experimental Results: Sparse Application

- Cluster of Pentium II 400MHz
  - Linux
  - 256MB main memory
  - 18GB local disk
  - Gigabit switch
- Total data of 2.7GB
  - Process about 1.9GB
  - Output 446MB
- 5 to 10 times faster
Experimental Results: Dense Application

- Multi-grid Virtual Microscope
  - Based on VMScope
  - Stores data on different resolutions
- Total data of 3.3GB
  - Process about 3GB
  - Output 1.6GB
- 2 to 3 times faster
Improving the Performance

- Virtual Microscope with subsampling
- Extra conditionals
  - From execution strategy
  - From application

```
for(i0 = low1; i0 <= hi1; i0++)
    for (i1 = low2; i1 <= hi2; i1++) {
        ipt[0] = i0;
        ipt[1] = i1;
        opt[0] = (i0-v0)/2;
        opt[1] = (i1-v1)/2;
        if ((tlow1 <= opt[0] <= thi1) &&
            (tlow2 <= opt[1] <= thi2))
            if ((i0 % 2 == 0) && (i1 % 2 == 0))
                O[opt].Accum(I[ipt]);
    }
```
Conditional Motion

- Eliminate redundant conditionals
- Views of a conditional
  - Syntactically different conditions
- Dominating constraints
  - Downward propagation
  - Upward propagation
- Omega Library
  - Generate code for a set of conditionals
for(i0 = low1; i0 <= hi1; i0++)
    for (i1 = low2; i1 <= hi2; i1++) {
        ipt[0] = i0;
        ipt[1] = i1;
        opt[0] = (i0-v0)/2;
        opt[1] = (i1-v1)/2;
        if ((tlow1 <= opt[0] <= thi1) &&
            (tlow2 <= opt[1] <= thi2))
            if ((i0 % 2 == 0) && (i1 % 2 == 0))
                O[opt].Accum(I[ipt]);
    }
if (2*low2 <= -v1+thi2 && low2 <= v1+2*thi2)
    for(t1 = max(2*(v0+2*tlow1+1)/2, 2*(low1+1)/2);
        t1 <= min(v0+2*thi1,hi1); t1+=2)
        for(t2 = max(2*(2*tlow2+va1+1)/2, 2*(low2+1)/2);
            t2 <= min(v1+2*thi2,hi2); t2+=2)
            s1(t1, t2);
\[ R := \{[i_0,i_1] : \text{low1} \leq i_0 \leq \text{hi1} \text{ and} \]
\[ \text{low2} \leq i_1 \leq \text{hi2} \text{ and} \]
\[ \exists i_{00}: i_{00} \times 2 = i_0 \text{ and} \]
\[ \exists i_{11}: i_{11} \times 2 = i_1 \} ; \]
\[ S := \{[i_0,i_1] : \text{tlow1} \times 2 + v_0 \leq i_0 \leq \text{thi1} \times 2 + v_0 \text{ and} \]
\[ \text{tlow2} \times 2 + v_1 \leq i_1 \leq \text{thi2} \times 2 + v_1 \} ; \]
\[ U := (R \text{ intersects } S) \]
\[ \text{codegen } U ; \]
Conditional Motion

subsampling vscope:

mg-vscope:

satellite:
Related Work

- Parallelizing irregular applications
  - Disk-resident datasets, different class of applications
- Out-of-core compilers
  - High-level abstractions, different applications, language, and runtime system
- Data-centric locality transformations
  - Focus on disk-resident datasets
- Synthesizing sparse applications from dense ones
  - Different class of applications, disk-resident datasets
- Code motion techniques
  - Target eliminating redundant conditionals
Conclusion

- High-level abstractions simplify application development
- Data-centric execution strategies help support efficient processing
- Data parallel framework is convenient to describe the applications
- Choice of strategies has substantial impact on the performance
Application Loops

Foreach \((r \in R)\) {
    \(O_1[S_{L1}(r)] = F_1(O_1[S_{L1}(r)], I_1[S_{R1}(r), ..., I_n[S_{Rn}(r)]];\)
    ...
    \(O_m[S_{Lm}(r)] = F_m(O_m[S_{Lm}(r)], I_1[S_{R1}(r), ..., I_n[S_{Rn}(r)]];\)
}

- Loop fission techniques to create canonical loops
- Program slicing techniques to extract the functions
Canonical Loops

- Facilitate the task for the run-time system
- Left hand side subscript functions
  - Output collections are congruent; or
  - All output collections fit in main memory
- Right hand side subscript functions
  - Input collections are congruent
  - $F_i(O_i, I_1, I_2, \ldots I_n) = g_0(O_i) \circled{op_1} g_1(I_1) \circled{op_2} g_2(I_2) \ldots \circled{op_n} g_n(I_n)$
  - $op_1$ to $op_n$ are commutative and associative
Program Slicing

public class VMPixel {
    char[3] colors;
    void Initialize()
    {
    }
    void Accum(VMPixel p, int avg) {
        colors[0] += p.colors[0]/avg;
        colors[1] += p.colors[1]/avg;
    }
}

public class VMPixelOut extends VMPixel implements Reducinterface;

public static void main(String[] args) {
    Point[2] lowend = [args[0], args[1]];
    RectDomain[2] querybox = [lowend:hiend];
    int subsamp = args[4];
    [[0,0]:((hiend-lowend)/subsamp];
    Point[2] p;
    foreach (p in OutDomain)
    {
        Output[p].Initialize();
    }
    foreach (p in querybox) {
        Point[2] q = (p - lowend)/subsamp;
        Output[q].Accum(Vscope[p], subsamp*subsamp);
    }
}

Public Class VMScope {
    static Point[2] lowpoint [0,0];
    static Point[2] hipoint[MaxX-1, MaxY-1];
    static RectDomain[2] VMSlide = [lowpoint:hipoint];
    static VMPixel[2d] Vscope = new VMPixel[VMSlide];
}