Domain Specific Languages
or
High Performance Computing for Dummies

Arun Chauhan
(*work with Ken Kennedy*)

Pizza Talk, Dept of Computer Science, Rice University, April 17th, 2000

What is Domain Specific Language?

- Language that enables problems in a specific domain to be described succinctly and easily
- Supporting libraries to enable high level operations to be treated as primitives
- E.g., Matlab®

Motivation

- Mathematics is the language of engineers and scientists
- They don't want to think in terms of variables, types, declarations, loops, caches, parentheses!
- But, there is a shortage of programmers

Existing Technology

- High level scripting languages, like Matlab®, Mathematica®, etc. provide Domain Specific Libraries and primitive operations
- Great for writing small programs
- Try simulating an airplane or weather forecasting model

Challenges

- **Understanding the specification**
  - inferencing variable types
  - dynamically varying types
- **High-level or source-level transformations**
  - domain-specific identities
- **Handling libraries** (and low-level transformations)
  - don't treat libraries as black-boxes!
  - apply low level transformations in application context

Pre-conditions

- Assume domain specific languages exist
- Programmer's Time is more expensive than CPU time
- Willing to compile scripts, as long as compilation is quick
- Willing to spend large amounts of time on compiling libraries
- High Performance the major objective for generated code

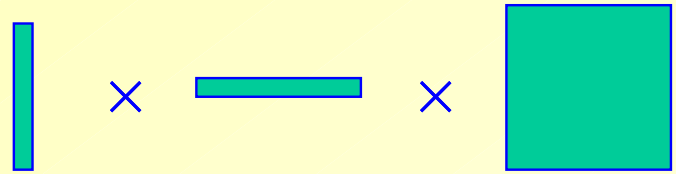
Understanding the Specification: First Step towards Compilation

- **Compiling requires, at least:**
  - type and shape inferencing at compile time
  - run-time resolution of ambiguities
- **State of the art**
  - commercial compilers, e.g., MCC, MATCOM
  - work by deRose and Padua (UIUC, 1995)
- **Interesting, but not the focus of our work**

High-Level Transformations

$$A \times B \times C = (A \times B) \times C = A \times (B \times C)$$

$A_{20\times1} \times B_{1\times20} \times C_{20\times20}$



$(A_{20\times1} \times B_{1\times20}) \times C_{20\times20}$ $\quad 20^2 + 20^3 = 8400$

$A_{20\times1} \times (B_{1\times20} \times C_{20\times20})$ $\quad 2 \times 20^2 = 800$

$$w = (A^T \times q) - (\beta \times w)$$

$w = (q^T \times A)^T - (\beta \times w)$ $\quad$ 24 times reduction in cost

What does it involve?

- **Vectorization of loops** (exposing high-level operations)

```
for i = 1:N
   xtemp = cos((i-1)*pi*x/L)
   for j = 1:N
      phi(k) = phi(k) + a(i,j)*xtemp*cos((j-1)*pi*y/L);
   end
end
```

```
xtemp_se = cos((0:N-1)*pi*x/L);
phi(k) = phi(k) + xtemp_se*a*cos((0:N-1)*pi*y/L)';
```

- **Utilizing domain-specific identities**
  - this also applies to library calls

Specifying Domain Specific Identities Using Formal Systems

Ajay Menon and Keshav Pingali (Cornell)

- **Define an abstract notation for matrices and matrix operations**
- **Write axioms to encode identities**
- **Convert code to the abstract notation and use axioms for transforming code**
- **But:**
  - **applied to a very limited domain**
  - **efficiency issues**
  - **interaction with low level transformations**

Transforming the Libraries

- Scripting language systems rely heavily on libraries
- Vast domain-specific libraries are available for domain-specific language systems
- Current compilers treat libraries as black boxes
- But, libraries are a key component of domain-specific language systems

Proposed Approach: Step 1

- **Compute promising optimizations in a library routine**
  - unit stride in matrix computations
  - special cases
  - shifted matrices
  - etc.

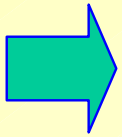
```
subroutine VMP (C, A, B, M, N, S)
  real A(N), B(N), C(M), S
  I = 1
  do J = 1, N
      C(I) = C(I) + A(J)*B(J)
      I = I + S
   enddo
end
```

  - example drivers
  - user annotations
  - self-learning AI techniques?

Proposed Approach: Step 2

- **Code specialization (partial evaluation)**

```
subroutine VMP (C, A, B, M, N, S)
  real A(N), B(N), C(M), S
  I = 1
  do J = 1, N
     C(I) = C(I) + A(J)*B(J)
     I = I + S
   enddo
end
```



S > 0

```
subroutine VMP (C, A, B, M, N, S)
  real A(N), B(N), C(M), S
  C(1:S*N-S+1:S) = C(1:S*N-S+1:S) + A(1:N)*B(1:N)
end
```

S = 0

```
subroutine VMP (C, A, B, M, N, S)
  real A(N), B(N), C(M), S
  C(1) = SUM(A(1:N)*B(1:N))
end
```

- trick is to recognize important conditions
- spend a lot of time tuning and optimizing library routines

Proposed Approach: Step 3

- **Construct efficient transfer functions**
  (also called jump functions)
    - function: actual parameters → side-effects

  - helps in rapidly choosing the right combination of specialized library routines
  - crucial to keep script compilation time low

Existing Technology

- **Well known techniques for whole program analysis**
- **Known techniques to compute transfer functions**
- **Automatic tuning of libraries for specific systems**
  - **ATLAS (Jack Dongarra, UTK)**

New Research

- Determining and propagating optimization conditions to call sites (library interfaces)
- Library annotation languages
  - one recently proposed by Samuel Guyer and Calvin Lin (UT Austin)
- Describing high level identities
- Tying high level transformations with low level transformations
- More ...

Conclusion

- Domain-Specific Scripting Languages are highly attractive for end users
- Current compiler technology provides many components needed for compiling scripts for high performance computation
- But, many open issues remain
- Scripts are also very relevant to GRID computation!!