
COMP 515: Advanced Compilation for Vector and Parallel Processors

Vivek Sarkar
Department of Computer Science
Rice University
vsarkar@rice.edu

<https://wiki.rice.edu/confluence/display/PARPROG/COMP515>

COMP 515

Lecture 19

8 November, 2011



Automatic Selection of High-Order Transformations in the IBM XL Fortran Compilers

Vivek Sarkar*

Rice University

`vsarkar@rice.edu`

* Work done at IBM during 1991 - 1993

High-Order Transformations

Traditional optimizations operate on a low-level intermediate representation that is close to the machine level

High-order transformations operate on a high-level intermediate representation that is close to the source level

Examples of high-order transformations: loop transformations, data alignment and padding, inline expansion of procedure calls, ...

Selection of High-Order Transformations

Improperly selected high-order transformations can degrade performance to levels worse than unoptimized code.

Traditional optimizations rarely degrade performance.

⇒ automatic selection has to be performed more carefully for high-order transformations than for traditional optimizations

This Work

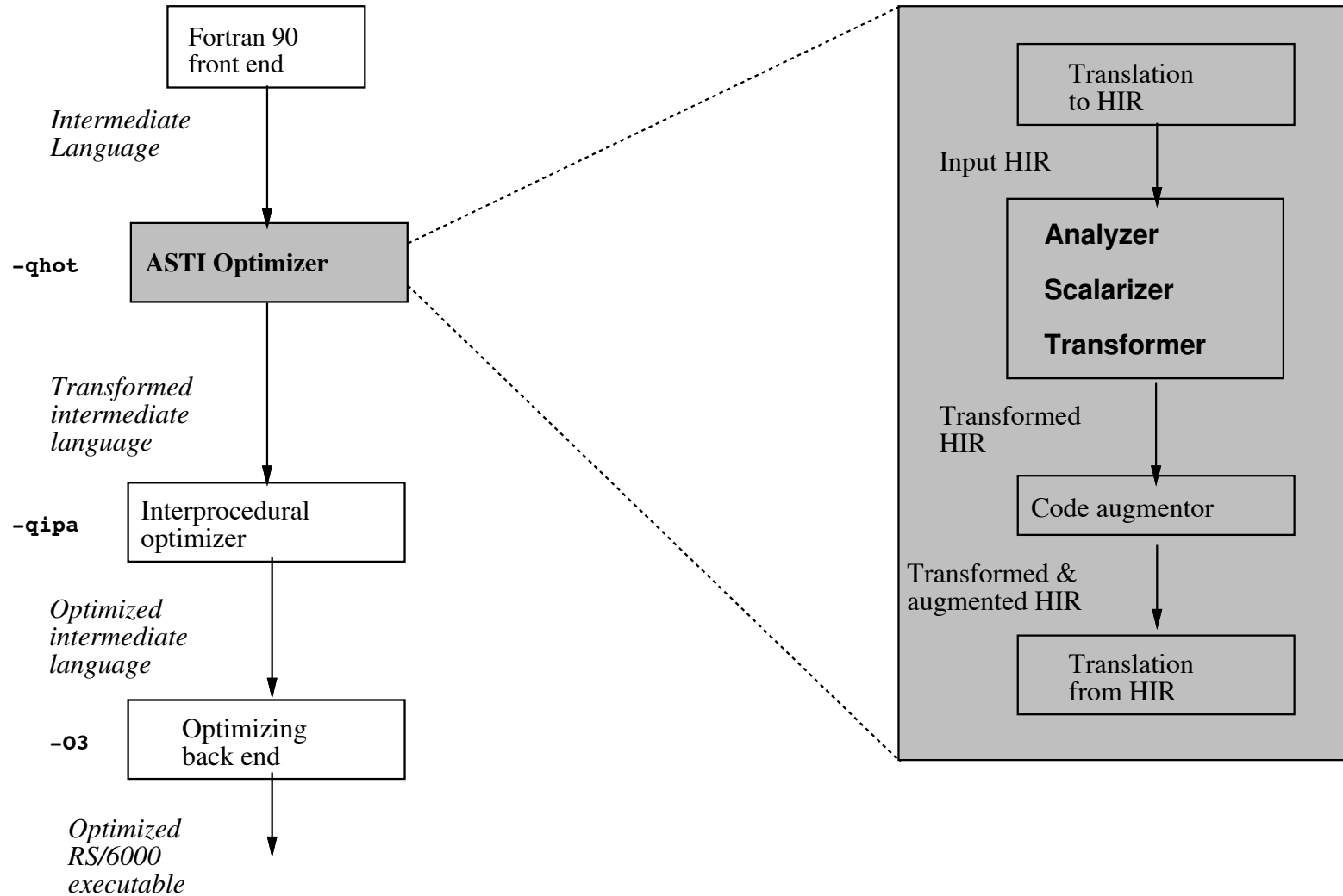
- Automatic selection of high-order transformations in the IBM XL Fortran compilers
- Quantitative approach to program optimization using cost models
- High-order transformations selected for *uniprocessor* target include: loop distribution, fusion, interchange, reversal, skewing, tiling, unrolling, and scalar replacement of array references
- Design and initial product implementation completed during 1991–1993

Reference: “Automatic Selection of High Order Transformations in the IBM XL Fortran Compilers”, V. Sarkar, IBM Journal of Res. & Dev., Vol. 41, No. 3, May 1997. (To appear).

Outline of Talk

1. Structure of compiler and optimizer
2. Memory cost analysis
3. Automatic selection of high-order transformations: loop interchange, loop tiling, loop fusion, scalar replacement, loop unrolling
4. Experimental results (preliminary)
5. Conclusions and future work

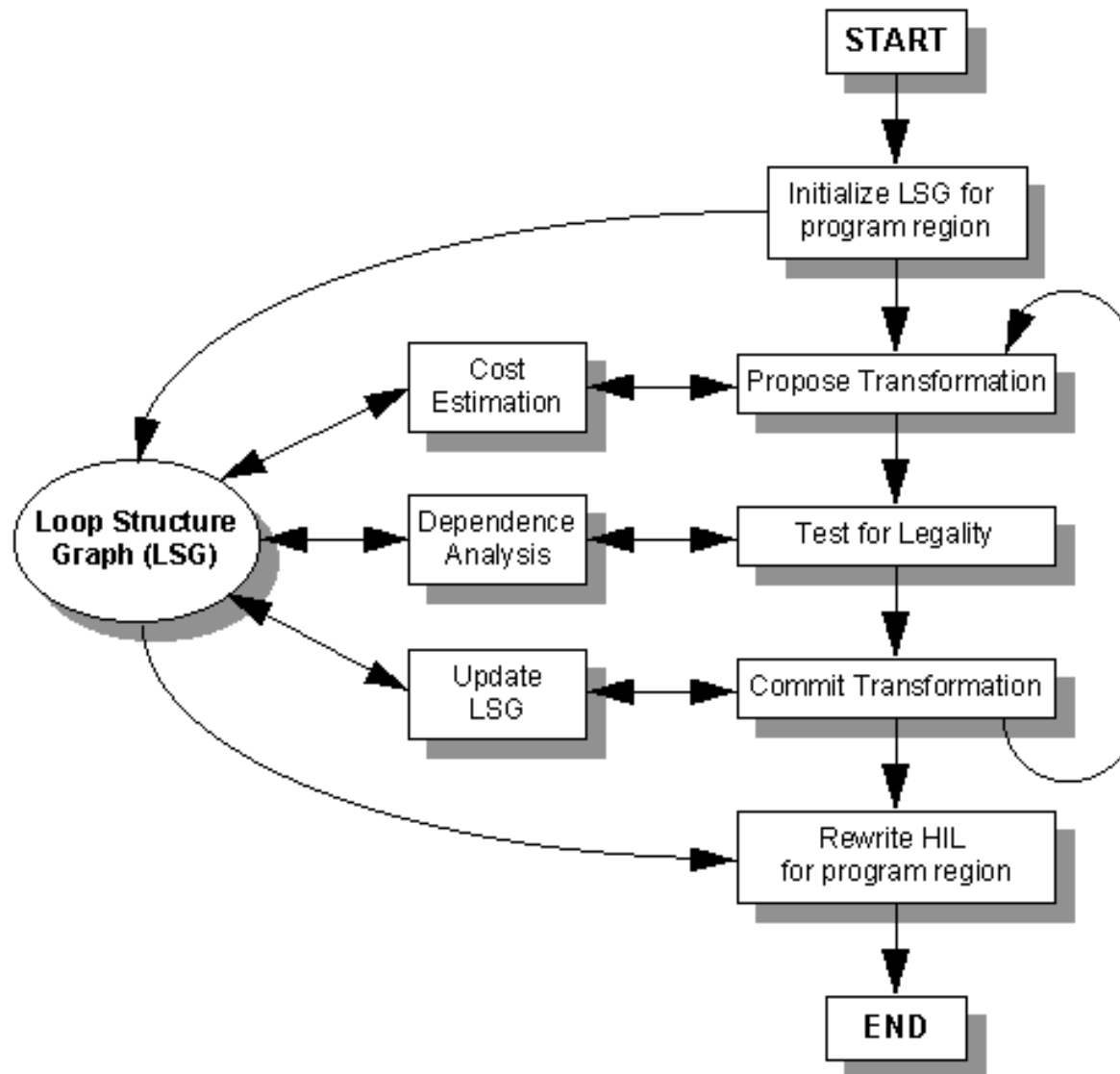
Structure of XL Fortran Product Compiler (Version 4)



Quantitative Approach to Program Optimization

- Compiler optimization is viewed as optimization problems based on quantitative cost models
- Cost models driven by compiler estimates of execution time costs, memory costs, execution frequencies (obtained either by compiler analysis or from execution profiles)
- Cost model depends on computer architecture and computer system parameters
- Individual program transformations used in different ways to satisfy different optimization goals

High level structure of the ASTI Transformer



Steps performed by ASTI Transformer

1. Initialization
2. Loop distribution
3. Identification of perfect loop nests
4. Reduction recognition
5. Locality optimization
6. Loop fusion
7. Loop-invariant scalar replacement
8. Loop unrolling and interleaving
9. Local scalar replacement
10. Transcription — generate transformed HIR

Memory Cost Analysis

Consider an innermost perfect nest of h loops:

```
do  $i_1 = \dots$   
  ...  
  do  $i_h = \dots$   
    ...  
  end do  
  ...  
end do
```

The job of memory cost analysis is to estimate

$DL_{total}(t_1, \dots, t_h) = \#$ distinct cache lines, and

$DP_{total}(t_1, \dots, t_h) = \#$ distinct pages

accessed by a (hypothetical) *tile* of $t_1 \times \dots \times t_h$ iterations.

Motivation for Memory Cost Functions

Assume that DL_{total} and DP_{total} are small enough so that no collision and capacity misses occur within a tile i.e.,

$$DL_{total}(t_1, \dots, t_h) \leq \text{effective cache size}$$

$$DP_{total}(t_1, \dots, t_h) \leq \text{effective TLB size}$$

The memory cost is then estimated as follows:

$$COST_{total} = (\text{cache miss penalty}) \times DL_{total} + (\text{TLB miss penalty}) \times DP_{total}$$

Our objective is to minimize the memory cost per iteration which is given by the ratio, $COST_{total}/(t_1 \times \dots \times t_h)$.

Matrix Multiply-Transpose Example

```
real*8 a(n,n), b(n,n), c(n,n)
```

```
. . .
```

```
do i1 = 1, n
```

```
  do i2 = 1, n
```

```
    do i3 = 1, n
```

```
      a(i1,i2) = a(i1,i2) + b(i2,i3) * c(i3,i1)
```

```
    end do
```

```
  end do
```

```
end do
```

Memory Cost Analysis for Matrix Multiply-Transpose Example

Assume cache line size, $L = 32$ bytes:

$$\begin{aligned}DL_{total}(t_1, t_2, t_3) &\approx \lceil 8t_1/L \rceil t_2 + \lceil 8t_2/L \rceil t_3 + \lceil 8t_3/L \rceil t_1 \\ &\approx (1 + 8(t_1 - 1)/L) t_2 + (1 + 8(t_2 - 1)/L) t_3 + \\ &\quad (1 + 8(t_3 - 1)/L) t_1 \\ &= (0.25t_1 + 0.75) t_2 + (0.25t_2 + 0.75) t_3 + \\ &\quad (0.25t_3 + 0.75) t_1\end{aligned}$$

Algorithm for selecting an optimized loop ordering

1. Build a symbolic expression for

$$F(t_1, \dots, t_h) = \frac{COST_{total}(t_1, \dots, t_h)}{t_1 \times \dots \times t_h}$$

2. Evaluate the h partial derivatives (slopes) of function F , $\delta F / \delta t_k$, at $(t_1, \dots, t_h) = (1, \dots, 1)$

A negative slope identifies a loop that carries temporal/spatial locality

3. Desired ordering is to place loop with most negative slope in innermost position, and so on.

Matrix Initialization example

```
do 10 i1 = 1, n
  do 10 i2 = 1, n
10      a(i1,i2) = 0
```

For a PowerPC 604 processor:

$$\begin{aligned}DL_{total}(t_1, t_2) &= (0.25t_1 + 0.75)t_2 \\DP_{total}(t_1, t_2) &= (0.001953t_1 + 0.998047)t_2 \\ \Rightarrow COST_{total}(t_1, t_2) &= 17 \times DL_{total}(t_1, t_2) + 21 \times DP_{total}(t_1, t_2) \\ &= (4.25t_1t_2 + 12.75t_2) + (0.04t_1t_2 + 20.96t_2) \\ \Rightarrow F(t_1, t_2) &= \frac{COST_{total}}{t_1t_2} = \left(4.25 + \frac{12.75}{t_1}\right) + \left(0.04 + \frac{20.96}{t_1}\right) \\ \Rightarrow \frac{\delta F}{\delta t_1} &= \frac{-33.71}{t_1^2} \text{ is } < 0 \text{ and } \frac{\delta F}{\delta t_2} = 0\end{aligned}$$

Desired loop ordering is i_2, i_1

Locality Analysis Approach

Progressive refinement of cache models:

- *Unbounded cache* – only *compulsory* misses
Solution: estimate # distinct accesses (DA)
- *Fully associative* with S lines/sets – also need to estimate *capacity* misses
Solution: adjust estimate by identifying *locality group*
- *Direct mapped* with S lines/sets – also need to estimate *collision* misses
Solution: further adjust estimate by computing cache utilization *efficiency* and *effective cache size*

Locality Group

Locality group – largest innermost iteration subspace that is guaranteed to incur no capacity or collision misses if execution is started with a clean/empty cache.

Locality group is specified by two parameters (m, B) :

1. $m \geq 0$, number of innermost loops in locality group.

$m = 0$ indicates that a single iteration overflows cache

2. B , largest number of iterations (block size) of the outermost loop in locality group.

B is only defined when $m \geq 1$

Using Locality Group to Estimate # Misses for a Fully-associative Cache

Summary of approach:

- Estimate # compulsory misses for single instance of locality group
- Ignore reuse among multiple instances of locality group
- Use pro-rated # misses/iteration from single instance to extrapolate to other instances of locality group

Set Conflicts and Effective Cache Size

```
DO 10 i = . . .  
10 A(T*i + c) = . . .
```

We are interested in estimating

$\eta(T)$ = cache utilization efficiency of stride T
= fraction of sets accessed over a large no. of iterations

Three cases of interest:

Case 1: $T \leq 2^b \Rightarrow \eta(T) = 1.0$

Case 2: T is a multiple of $2^b \Rightarrow \eta(T) = \frac{1}{\gcd(T/2^b, 2^s)}$

Set Conflicts and Effective Cache Size (contd.)

Case 3: Otherwise

Find smallest $n \geq 1$ that satisfies

$$\text{mod}(n \times T, 2^{b+s}) < 2^b \text{ or } \text{mod}(n \times T, 2^{b+s}) > 2^{b+s} - 2^b$$

\Rightarrow every n 'th access will map to the same set

$$\Rightarrow \eta(T) = \frac{n}{2^s}$$

NOTE: more precise estimates are possible if cache block alignment offset, number of loop iterations, and degree of cache associativity are also taken into account

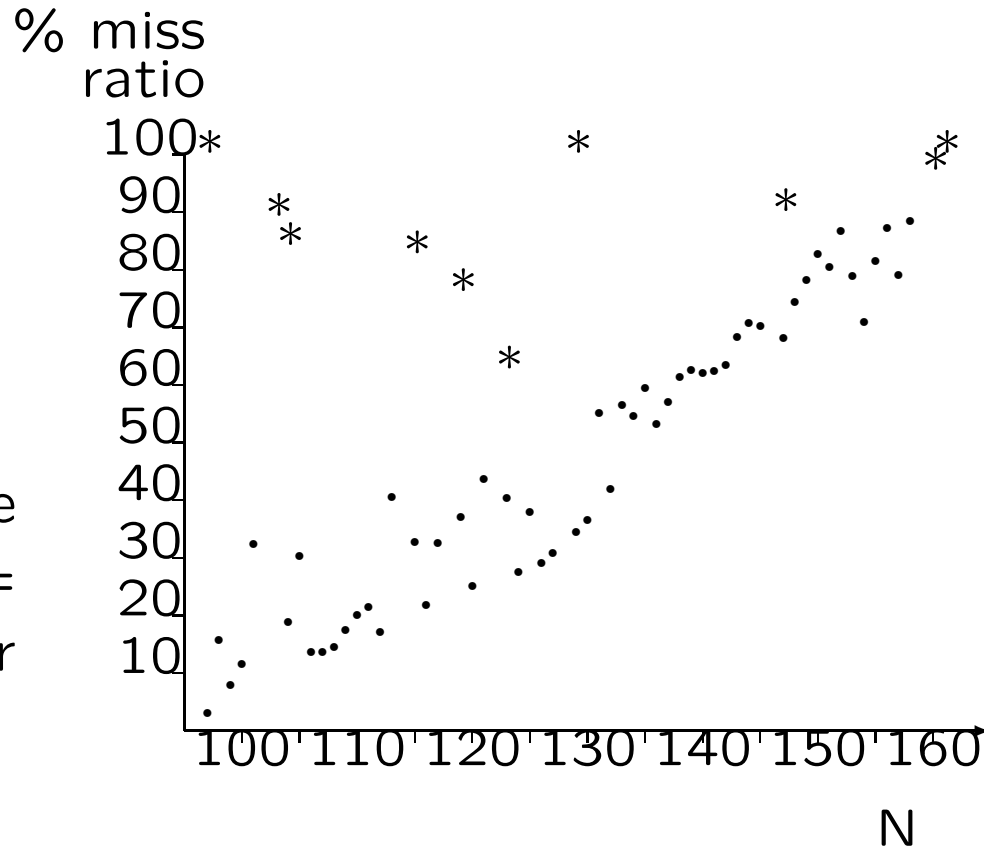
Example of Set Conflicts

```

real*8 A(N,N)
do 10 i = 1, N
  do 10 j = 1, N
    do 10 k = 1, N
10      . . . A(i,k)

```

Simulate A(i,k) reference for cache parameters, $2^b = 16$, $2^s = 32$, $2^d = 4$, and for $96 \leq N \leq 160$



Set conflict analysis identifies the main outlying points, $N = 96, 102, 103, 114, 118, 122, 128, 146, 159, 160$

Homework #6 (Written Assignment)

Read Section 6 (Memory Cost Analysis) of the following paper discussed in today's lecture, especially the partial derivative analysis on pg 15 (printed page 247):

• [Automatic Selection of High Order Transformations in the IBM XL Fortran Compilers](#). Vivek Sarkar. IBM Journal of Research and Development, 41(3), May 1997

1. Compute the memory cost function and partial derivatives for loops I and J in the following loop nest at the start of Section 9.3.5 of the course textbook. Which loops carry locality? Can all of them be moved to the innermost position?

```
DO I = 1, N
  DO J = 1, M
    A(J+1) = (A(J)+A(J+1))/2
  ENDDO
ENDDO
```

2. Compute the memory cost and partial derivatives for loops I and J in the following transformed loop nest (after skewing) in Section 9.3.5 of the course textbook. Which loops carry locality? Can all of them be moved to the innermost position?

```
DO I = 1, N
  DO j = I, M + I - 1
    A(j-I+2) = (A(j-I+1)+A(j-I+2))/2
  ENDDO
ENDDO
```

Homework #6 (contd)

- Due by 5pm on Tuesday, November 15th
- Homework should be turned into Amanda Nokleby, Duncan Hall 3137
- Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates, the teaching assistants and the professor, but you should never misrepresent someone else's work as your own. If you use any material from external sources, you must provide proper attribution.