COMP 515: Advanced Compilation for Vector and Parallel Processors

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

https://wiki.rice.edu/confluence/display/PARPROG/COMP515
Homework #3 (REMINDER)

1. Solve exercise 5.6 in book
   — Your solution should be legal for all values of \( K \) (note that the value of \( K \) is invariant in loop I)

Exercise 5.6: What vector code should be generated for the following loop?

DO I = 1, 100
   A(I) = B(K) + C(I)
   B(I+1) = A(I) + D(I)
END DO

• Due on Oct 8th
Recap

• More transformations to expose more fine-grained parallelism
  — Node Splitting
  — Recognition of Reductions
  — Index-Set Splitting
  — Run-time Symbolic Resolution
  — Loop Skewing

Previous lecture

• Unified framework to generate vector code

This lecture

• Note: these transformations are useful for generating other forms of parallel code as well (beyond vector)
Run-time Symbolic Resolution

• “Breaking Conditions”

DO I = 1, N
   A(I+L) = A(I) + B(I)
ENDDO

Transformed to..

IF(L .LE. 0 .OR. L .GT. N) THEN
   A(L+1:N+L) = A(1:N) + B(1:N)
ELSE
   DO I = 1, N
      A(I+L) = A(I) + B(I)
   ENDDO
ENDIF
Run-time Symbolic Resolution

• Identifying minimum number of breaking conditions to break a recurrence is NP-hard
  — NOTE: in practice, this can be more important for conditions related to pointer aliasing than for array subscripts

• Heuristic:
  — Identify when a critical dependence can be conditionally eliminated via a breaking condition
Loop Skewing

• Reshape Iteration Space to uncover parallelism

```
DO I = 1, N
    DO J = 1, N
        (=,<)
        S: A(I,J) = A(I-1,J) + A(I,J-1)
        (<,=)
    ENDDO
ENDDO
```

• Parallelism not apparent at loop level, and interchange doesn’t help
Loop Skewing

- Dependence Pattern before loop skewing
Loop Skewing

• Do the following transformation called loop skewing

\[ jj = J + I \text{ or } J = jj - I \]

\[
\begin{align*}
\text{DO } I &= 1, N \\
\text{DO } jj &= I+1, I+N \\
    J &= jj - I \\
\text{S: } A(I, J) &= A(I-1, J) + A(I, J-1)
\end{align*}
\]

Note: Direction Vector Changes, but statement body remains the same
(Examples in textbook usually copy propagate \( J = jj - I \) in all uses of \( J \))
Loop Skewing

• Dependence pattern after loop skewing
  —NOTE: Replace j by jj in figure below
Loop Skewing

DO I = 1, N  ! DV = { (<,<), (=, <) }
    DO jj = I+1, I+N
    ENDDO
ENDDO

Loop interchange to..
DO jj = 2, N+N  ! DV = { (<,<), (<, =) }
    DO I = max(1,jj-N), min(N,jj-1)
    ENDDO
ENDDO

Vectorize to..
DO jj = 2, N+N
    FORALL I = max(1,jj-N), min(N,jj-1)
    END FORALL
ENDDO
Loop Skewing

- Disadvantages:
  - Varying vector length
    - Not profitable if $N$ is small
  - If vector startup time is more than speedup time, this is not profitable
  - Vector bounds must be recomputed on each iteration of outer loop

- Apply loop skewing if everything else fails

- We will later study Unimodular and Polyhedral transformations, which include generalizations of loop skewing
Chapter 5: Putting It All Together

• **Good Part**
  — Many transformations imply more choices to exploit parallelism

• **Bad Part**
  — Choosing the right transformation
  — How to automate transformation selection process?
  — Interference between transformations
Putting It All Together

• Example of Interference

DO I = 1, N
  DO J = 1, M
    S(I) = S(I) + A(I,J)
  ENDDO
ENDDO

*Sum Reduction gives..*

DO I = 1, N
  S(I) = S(I) + SUM (A(I,1:M))
ENDDO

*While Loop Interchange and Vectorization gives..*

DO J = 1, N
  S(1:N) = S(1:N) + A(1:N,J)
ENDDO
Putting It All Together

- Any algorithm which tries to tie all transformations must
  - Take a global view of transformed code
  - Know the architecture of the target machine

- Goal of our algorithm
  - Finding ONE good vector loop in each loop nest [works well for most
    vector register architectures]
Unified Framework

- Detection: finding ALL loops for EACH statement that can be run in vector
- Selection: choosing best loop for vector execution for EACH statement
- Transformation: carrying out the transformations necessary to vectorize the selected loop

- See Section 5.10 for details
## Performance on Benchmarks

<table>
<thead>
<tr>
<th>Vectorizing Compiler</th>
<th>Total</th>
<th>Dependence</th>
<th>Vectorization</th>
<th>Idioms</th>
<th>Completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V</td>
<td>P</td>
<td>N</td>
<td>V</td>
<td>P</td>
</tr>
<tr>
<td>PFC</td>
<td>70</td>
<td>6</td>
<td>24</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>Alliant FX/8, Fortran V4.0</td>
<td>68</td>
<td>5</td>
<td>27</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>Amdahl VP-E, Fortran 77</td>
<td>62</td>
<td>11</td>
<td>27</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>Ardent Titan-1</td>
<td>62</td>
<td>6</td>
<td>32</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>CDC Cyber 205, VAST-2</td>
<td>62</td>
<td>5</td>
<td>33</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>CDC Cyber 990E/995E</td>
<td>25</td>
<td>11</td>
<td>64</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Convex C Series, FC 5.0</td>
<td>69</td>
<td>5</td>
<td>26</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>Cray series, CF77 V3.0</td>
<td>69</td>
<td>3</td>
<td>28</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>CRAX X-MP, CFT V1.15</td>
<td>50</td>
<td>1</td>
<td>49</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Cray Series, CFT77 V3.0</td>
<td>50</td>
<td>1</td>
<td>49</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>CRAY-2, CFT2 V3.1a</td>
<td>27</td>
<td>1</td>
<td>72</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>ETA-10, FTN 77 V1.0</td>
<td>62</td>
<td>7</td>
<td>31</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>Gould NP1, GCF 2.0</td>
<td>60</td>
<td>7</td>
<td>33</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>Hitachi S-810/820,</td>
<td>67</td>
<td>4</td>
<td>29</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>IBM 3090/VF, VS Fortran</td>
<td>52</td>
<td>4</td>
<td>44</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Intel iPSC/2-VX, VAST-2</td>
<td>56</td>
<td>8</td>
<td>36</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>NEC SX/2, F77/SX</td>
<td>66</td>
<td>5</td>
<td>29</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>SCS-40, CFT x13g</td>
<td>24</td>
<td>1</td>
<td>75</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Stellar GS 1000, F77</td>
<td>48</td>
<td>11</td>
<td>41</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>Unisys ISP, UFTN 4.1.2</td>
<td>67</td>
<td>13</td>
<td>20</td>
<td>21</td>
<td>3</td>
</tr>
</tbody>
</table>

16 PFC = Parallel Fortran Converter tool developed at Rice by Allen & Kennedy
Test 171: One example that PFC was unable to vectorize

DO I = 1, N
    A(I*N) = A(I*N) + B(I)
ENDDO
Coarse-Grain Parallelism

Chapter 6 of Allen and Kennedy
Introduction

• Previously, our transformations targeted vector and superscalar architectures.

• In Chapter 6, we worry about transformations for symmetric multiprocessor (multicore) machines.

• The difference between these transformations tends to be one of granularity.
Review

• SMP machines have multiple processors all accessing a central memory.

• The processors are unrelated, and can run separate processes.

• Starting processes and synchronization between processes is expensive.
Synchronization

• A basic synchronization element is the barrier at the end of a parallel loop.
• A barrier in a program forces all processes to reach a certain point before execution continues.
• Bus contention can cause slowdowns.
Techniques for parallelizing a single loop

- Single loop methods
  - Privatization
  - Loop distribution
  - Loop fusion
  - Alignment
  - Code replication
Single Loops

- The analog of scalar expansion is privatization.
- Temporaries can be given separate namespaces for each iteration.

```
DO I = 1,N
S1     T = A(I)
S2     A(I) = B(I)
S3     B(I) = T
ENDDO

PARALLEL DO I = 1,N
PRIVATE t
S1    t = A(I)
S2    A(I) = B(I)
S3    B(I) = t
ENDDO
```
Definition: A scalar variable $x$ in a loop $L$ is said to be privatizable if every path from the loop entry to a use of $x$ inside the loop passes through a definition of $x$.

Privatizability can be stated as a data-flow problem:

$$
up(x) = use(x) \cup (!\text{def}(x) \cap \bigcup_{y \in \text{succ}(x)} up(y))
$$

$$
\text{private}(L) = \neg up(\text{entry}) \cap (\bigcup_{y \in L} \text{def}(y))
$$

We can also do this by declaring a variable $x$ private if its SSA graph doesn’t contain a phi function at the entry.
Course Schedule

- No class on October 1 and October 8
- Individual project meetings will be scheduled during Oct 12-13