Hybrid Analysis
and its Application to Thread Level Parallelization

Lawrence Rauchwerger

Parasol
Smarter computing.
Texas A&M University
Thread (Loop) Level Parallelization

- Thread level Parallelization
  - Extracting parallel threads from a sequential program
- Manual
  - Very expensive for sequential legacy code: MPI - OpenMP

- Automatic
  - So far not very successful due to:
    - weak static analysis
    - Decisions depend on dynamic values (input dependent)
What is a parallel loop?

**Parallelizable loop**

\[
\text{DO } i = 1, 100 \\
a[i] = a[i] + b[i] \\
\text{ENDDO}
\]

**Sequential loop**

\[
\text{DO } i = 1, 100 \\
a[i+1] = a[i] + b[i] \\
\text{ENDDO}
\]

**Flow, i \rightarrow i+1**

<table>
<thead>
<tr>
<th>i=1</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=2</td>
<td>RW</td>
</tr>
<tr>
<td>i=3</td>
<td>RW</td>
</tr>
<tr>
<td>i=4</td>
<td>RW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i=1</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=2</td>
<td>RW</td>
</tr>
<tr>
<td>i=3</td>
<td>RW</td>
</tr>
<tr>
<td>i=4</td>
<td>RW</td>
</tr>
</tbody>
</table>
Static Data Dependence Analysis: Linear Reference Patterns

- Geometric view: Polytope model
  - Some convex body contains no integral points
- Simplified solutions: GCD Test, Banerjee Test etc
  - Potentially overly conservative
- General solution: Presburger formula decidability
  - Omega Test: Precise, potentially slow

DO j = 1, 10
a(j) = a(j+40)
ENDDO

No cross iteration dependences
No integer solutions:

\[ 1 \leq j_w \leq 10 \]
\[ 1 \leq j_r \leq 10 \]
\[ j_w \neq j_r \]
\[ j_w = j_r + 40 \]

Restricted to linear addressing and control: mostly small kernels
Static Data Dependence Analysis: Nonlinear Reference Patterns

No known solution in general

Common cases:
- Indirect access (subscripted subscripts)
- Control dependence on data values
- Recurrence without closed form

```
DO j = 1, 10
  IF (x(j) > 0) THEN
    a(f(j)) = ...
  ENDIF
ENDDO
```

- Linear Approximation
  - Maslov ’94; Creusillet, Irigoin ‘96
  - Range Test (Blume, Eigenmann ‘94)
  - Monotonicity (Wu, Cohen, Padua ’01)

- Symbolic Unknowns
  - Uninterpreted Function Symbols (Pugh, Wonnacott ’95)
  - Fuzzy Array Dataflow (Barthou, Collard, Feautrier ’95)

- User assistance
  - Uninterpreted Function Symbols (Pugh, Wonnacott ’95)
  - SUIF Explorer (Liao ’00)

Most nonlinear cases not solvable statically: need run time analysis
Alternative: Run-time Analysis

```fortran
READ *, N
DO j=1,N
   a(j)=a(j+40)
ENDDO
```

Solution: **LRPD Run-time Test** (Rauchwerger and Padua '95)

- Instrument all relevant memory references
- Analyze the resulting trace at run time

### Examples

- **N = 5**
  - READ
  - WRITE

### Diagram

- **N = 5**
  - Independent
  - N = 41, 45

- **N = 45**
  - Dependent
  - Accurate, but overhead proportional to the dynamic memory reference count

Minimum necessary information for parallelization: \( N \leq 40 \)

Linear, very simple, but not decidable statically!
Compile-time vs. Run-time

**Compile Time**

- **PROs**
  - No run-time overhead

- **CONs**: too conservative when
  - Input/computed values
    - Indirection
    - Control
  - Weak symbolic analysis
    - Complex recurrences
  - Impractical symbolic analysis
    - Combinatorial explosion

**Run-time, reference by reference**

- **PROs**
  - Always finds answers

- **CONs**
  - Run-time overhead proportional to dynamic memory reference count
  - Unnecessary work: Ignores partial compile-time results
Why Did We Fail?

- Static Analysis cannot be sufficient (weak and/or input sensitive)
- Dynamic Analysis (misunderstood as only speculation) is not a substitute for Static Analysis (too costly)
- No program level representation suitable for representing statically un-analyzable code
Hybrid Analysis of memory Reference Patterns (Rus ICS’02, 07)

<table>
<thead>
<tr>
<th></th>
<th>Compile-time Analysis</th>
<th>Hybrid Analysis</th>
<th>Run-time Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATIC</td>
<td>Symbolic analysis</td>
<td>Symbolic analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extract conditions</td>
<td></td>
</tr>
<tr>
<td>DYNAMIC</td>
<td></td>
<td>Evaluate conditions</td>
<td>Full reference-by-reference analysis</td>
</tr>
</tbody>
</table>

**Framework**: Hybrid memory reference analysis

**Application**: Automatic parallelization
Hybrid Analysis

4. a) If we can prove $10 \leq N \leq 30$, generate parallel loop.

4. b) If $N$ is unknown, Extract run-time test.

Compile Time

Run Time

Parallel Loop

DO PARALLEL j=1,N
  a(j)=a(j+40)
ENDDO

No run-time tests performed if not necessary!

Parallel Loop

DO j=1,N
  a(j)=a(j+40)
ENDDO

Sequential Loop

IF ($N \leq 40$) THEN
  DO PARALLEL j=1,N
    a(j)=a(j+40)
  ENDDO
ELSE
  DO j=1,N
    a(j)=a(j+40)
  ENDDO
ENDIF

Run-time Test
Under what conditions can the loop be executed in parallel?

1. Collect and classify memory references.
2. Aggregate them symbolically.
3. Formulate independence test.

4.a) If we can prove $10 \leq N \leq 30$, Declare loop parallel.

4.b) If $N$ is unknown, Extract run-time test.
Aggregation of Linear References Across an Iteration Space

WF pattern for $A$:

```
DO j=1,100
  A(j) = ...
ENDDO
```

$1:100$

LMAD = Linear Memory Access Descriptor (Hoeflinger ’98)
Multidimensional, strided intervals
Gate Operator
Postpone Analysis Failure

The truth value of \((x=0)\) is not known
Recurrence Operator
Postpone Static Analysis Failure due to Nonlinear Reference Pattern

Recurrence with no closed form on $x$
Uniform Set of References (USR)

Closed under composition at program level

\[ T = \{ \text{LMAD}, \cap, \cup, -, (, ), \#, \otimes, \Theta, \text{Gate, Recurrence, Call Site} \} \]

\[ N = \{ \text{USR} \} \]

\[ S = \{ \text{USR} \rightarrow \text{LMAD} \mid (\text{USR}) \} \]

\[ P = \{ \text{USR} \rightarrow \text{USR} \cup \text{USR} \]
\[ \text{USR} \rightarrow \text{USR} \cap \text{USR} \]
\[ \text{USR} \rightarrow \text{USR} - \text{USR} \]
\[ \text{USR} \rightarrow \text{Gate} \# \text{USR} \]
\[ \text{USR} \rightarrow \text{Recurrence} \otimes \text{USR} \]
\[ \text{USR} \rightarrow \text{Call Site} \Theta \text{USR} \} \]

\[ \text{LMAD} = \text{Start} + [\text{Stride}_1: \text{Span}_1, \text{Stride}_2: \text{Span}_2, \ldots] \]
Are there any cross-iteration dependences?

1. Collect references.

2. Aggregate them symbolically & Classify Reference Type

3. Formulate independence test.

4. Extract lowest-cost runtime test.

N ≤ 40
Data Dependences

- Given:
  - Loop expression: \( j = 1, N \)
  - Per-iteration aggregated descriptors \( \text{RO}_j, \text{WF}_j, \text{RW}_j \)

- Solve equation \( \text{RO} \cap \text{WF} = \emptyset \)

- At compile-time:
  - \( \text{RO} \cap \text{WF} \) evaluates to \( \emptyset \) \( \Rightarrow \) independent
  - \( \text{RO} \cap \text{WF} \) evaluates to a set that is not empty \( \Rightarrow \) dependent
  - All other cases: run-time dependence test

Similar equations for privatization and reduction recognition
Are there any cross-iteration dependences?

1. Collect references.

2. Aggregate them symbolically & Classify Reference Type

3. Formulate independence test.

4. Extract lowest-cost runtime test.

N ≤ 40
Algorithm: Recursive Descent

DO j = 1, n
  A(j) = A(j+40)
  IF (x>0) THEN
    A(j) = A(j) + A(j+20)
  ENDIF
ENDDO

1. Distribute Intersection

Empty? Empty?

1:n # 41:40+n

21:20+n x>0

Empty? Empty?

1:n # 41:40+n 1:n

Empty? Empty?

2

1:n

Empty? Empty?

4

(n ≤ 20 or x ≤ 0) and n ≤ 40

Empty? Empty?

4

1:n

(21:20+n)

Empty? Empty?

3

n ≤ 40

n ≤ 20 x ≤ 0

Empty? Empty?

3

x ≤ 0

n ≤ 40

21:20+n 1:n
Novel Static/Dynamic Interface: Predicate DAG

Represents any dynamic condition for loop parallelization

\[ T = \{ \text{Logical Expression, } \land, \lor, \land, \lor, \Theta, \text{Recurrence, Call Site, Library Routine} \} \]

\[ N = \{ \text{PDAG} \} \]

\[ S = \text{PDAG} \]

\[ P = \{ \text{PDAG} \rightarrow \text{Logical Expression} \]
\[ \text{PDAG} \rightarrow \text{PDAG} \land \text{PDAG} \]
\[ \text{PDAG} \rightarrow \text{PDAG} \lor \text{PDAG} \]
\[ \text{PDAG} \rightarrow \text{PDAG} \times \land \text{Recurrence} \]
\[ \text{PDAG} \rightarrow \text{PDAG} \times \lor \text{Recurrence} \]
\[ \text{PDAG} \rightarrow \text{PDAG} \Theta \text{Call Site} \]
\[ \text{PDAG} \rightarrow \text{Library Routine} \} \]

- **Expressive**: arbitrary dependence questions
- **Inexpensive**: evaluates quickly at run-time
PDAG Extraction

Input: USR equation $D = \emptyset$

Output: PDAG $P$

Such that:

$P \iff D = \emptyset$

Optimistic: Sufficient predicate: $P \Rightarrow D = \emptyset$

Pessimistic: Necessary predicate: $D = \emptyset \Rightarrow P$, or $\overline{P} \Rightarrow D \neq \emptyset$
Fallback

Not all equations can be reversed

\[ \text{Empty?} \]

\[ \bigcap \]

\[ \times \]

\[ \text{ind}_1(j) \]

\[ j=1,100 \]

\[ \times \]

\[ \text{ind}_2(j) \]

\[ j=1,100 \]

Fallback Solutions

Trace based

Speculation

Possibly Aggregated

\[
\begin{align*}
\text{DO } & j=1,100 \\
A(\text{ind}_1(j)) & = A(\text{ind}_2(j)) \\
\text{ENDDO}
\end{align*}
\]
Hybrid Analysis: Run Time

- O(1) Scalar Comparison: Fail
- O(n/k) Comparisons: Fail
- Reference Based: Fail

Independent

Dependent
Evaluation Methodology

• Automatic parallelization using Hybrid Analysis in Polaris
  • Analyses/Transformations: Dependence analysis, privatization, reduction, pushback
  • Candidate loop selection: Profiling
  • No scheduling or memory locality optimization
  • Simple dynamic mechanism to rule out very small loops

Fortran 77 → Polaris HA → Fortran 77 + OpenMP → Intel ifort version 9.0 → Executable Code

• Experiment Setup
  • SPEC 2000 - 2006, PERFECT, Other SPEC
  • Dual Core CoreDuo Intel, Lenovo X60s
  • Dual Core AMD, quad socket Sun
Polaris-HA vs. Ifort Coverage: PERFECT/SPEC89/92
Polaris-HA vs. Ifort Coverage: PERFECT Benchmarks
Polaris-HA vs. Ifort Coverage: SPEC 89/92
Polaris-HA vs. Ifort Coverage: SPEC2000/06
Speedups SPEC2000/06: AMD dual core/quad socket
Speedups Perfect/Spec89/92: Intel Core Duo
Speedups PERFECT: Intel CoreDuo
Speedups SPEC 89/92: Intel Core Duo
Conclusions: HA + Autopar

- Hybrid memory reference analysis is a general framework for optimization
- Representation is crucial
  -- USR
    - Closed-form representation that tolerates analysis failure
  - PDAG:
    - Input sensitivity of optimization decisions
    - Continuum of compile-time to run-time solutions
- Efficient automatic parallelization
  - Good Speedups on FP benchmark applications

http://parasol.tamu.edu/compilers/ha