The Challenges of Non-linear Parameters and Variables in Automatic Loop Parallelisation

Armin Größlinger
December 2, 2009

Rigorosum

Fakultät für Informatik und Mathematik
Universität Passau
Automatic Loop Parallelisation

for (i=1; i<=n; i++)
  for (j=1; j<=n-i; j++)

for (t=1; t<=n; t++)
  parfor (p=1; p<=t; p++)
    A[t-p+1][p] = ...;

Transformation(s)

Code
generation

Analysis

1 ≤ i ≤ n
1 ≤ j ≤ n − i
Dependences:
(i, j) → (i+1, j)
(i, j) → (i, j+1)

Loop bounds and array indices are linear (affine) expressions.

Polyhedron model

1 ≤ t ≤ n
1 ≤ p ≤ t
(t, p) → (t+1, p)
(t, p) → (t+1, p+1)
Non-linearity?

The polyhedron model can handle some codes in, e.g.,
- Simulation, image processing, linear algebra.

Today, parallelism is everywhere:
- Multi-core CPUs, many-core CPUs, graphics card computing (GPGPU)
- Automatic parallelisation helps not to burden software developers with the parallelism.
- Non-linearities make the polyhedron model more widely applicable:
  - Handle more programs,
  - Target more diverse hardware.
Non-linearity

- Linear: \( A[2i + 3j - 4m + 5n + 7] \) expressions linear in the variables \textit{and} the parameters.

- Non-linearity:
  - \( A[n*i + m*m*j + n*m] \)
    Expressions still linear in the variables ("non-linear parameters").
  - \( A[i*j + m*j*j] \)
    Arbitrary polynomials in the variables and parameters.
for (i=1; i<=n; i++)
  for (j=1; j<=n-i; j++)
  ...
Dependence Analysis Example

for (i=0; i<=m; i++)
  for (j=0; j<=m; j++)
    ... A[p*i+2*j] ...

"When is A[x] accessed again?"

Which iterations (i,j) access the same array element?

Result of our automatic analysis:

(i,j) → (i + 1, j − \frac{p}{2}) \text{ if } \begin{cases} p \equiv 2 \mod 0, m \geq 1, -2m \leq p \leq 2m, 0 \leq i \leq m-1, \\
\max(0, \frac{p}{2}) \leq j \leq \min(m, m+\frac{p}{2}) \end{cases}

(i,j) → (i + 2, j − p) \text{ if } \begin{cases} p \equiv 2 \mod 1, m \geq 2, -m \leq p \leq m, 0 \leq i \leq m-2, \\
\max(0, p) \leq j \leq \min(m, m+p) \end{cases}

(Trying to use weak quantifier elimination in the integers to compute the dependences yields an output with > 20,000 lines.)
for (i=0; i<=m; i++) {
    for (j=0; j<=n; j++) {
        ... A[4*i+2*j] ...
    }
    ... A[p*i+1] ...
}

4 \cdot i + 2 \cdot j = p \cdot i' + 1

\begin{pmatrix}
  4 \\
  2 \\
  -p
\end{pmatrix} = 1

Solutions for \( i, j, i' \in \mathbb{Z} \) in dependence of \( p \in \mathbb{Z} \)?

For \( p \equiv 2 \ 0 \): no solution.

For \( p \equiv 2 \ 1 \): 

\[
i = t_1
\]

\[
 j = (-2p - 2) \cdot t_1 - p \cdot t_2 - \frac{p + 1}{2}
\]

\[
i' = -4t_1 - 2t_2 + 1 \quad \text{for } t_1, t_2 \in \mathbb{Z}.
\]
Linear Diophantine Equation Systems

To solve a system of linear Diophantine equations
\[ x \cdot A = b \]
with \( A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^n \)
for \( x \in \mathbb{Z}^m \), all we need is an algorithm to compute GCDs.
(More precisely, for \( c, d \in \mathbb{Z} \), we must be able to compute 
\( g, u, v \in \mathbb{Z} \) such that: \( \gcd \mathbb{Z}(c, d) = g = u \cdot c + v \cdot d \).)

Result: We can perform a similar procedure when \( A \) and \( b \)
depend on \( p \in \mathbb{Z} \), i.e., we want to solve
\[ x \cdot A(p) = b(p) \]
for \( x \) in dependence of \( p \).

Generalisation

How do we generalise the classical procedure to solve

\[
(i \ j \ i') \begin{pmatrix}
4 \\
2 \\
-p
\end{pmatrix} = 1
\]

What is the GCD of 2 and \( p \)?

\[
\gcd_{\mathbb{Z}}(2, p) = \begin{cases}
2 & \text{if } p \equiv 2 \pmod{2} \\
1 & \text{if } p \equiv 2 \pmod{1}
\end{cases}
\]

Modelling \( p \) by the unknown \( X \) of \( \mathbb{Z}[X] \) does not work:

\[
\gcd_{\mathbb{Z}[X]}(X, 2) = 1
\]

\[
\gcd_{\mathbb{Z}[X]}(f, g)(p) \neq \gcd_{\mathbb{Z}}(f(p), g(p)) \quad \text{(in general)}
\]

“polynomial GCD” \hspace{1cm} “pointwise GCD”

Is there a polynomial ring \( \mathbb{Z}[X] \subseteq \mathbb{Z}[X] \) in which polynomial and pointwise GCD coincide?
**Entire Quasi-polynomials**

**Definition.** A function $c : \mathbb{Z} \to \mathbb{Q}$ with period $l \geq 1$, i.e., $\forall p \in \mathbb{Z} : c(p) = c(p + l)$ is called a *periodic number.*

Notation: $[c(0), \ldots, c(l - 1)]$, e.g., $[1, 2, 3]$.

**Definition.** $f = \sum_{i=0}^{u} c_i X^i$ with periodic numbers $c_i$ as coefficients is called a *quasi-polynomial.*

**Evaluation:** $f(p) := \sum_{i=0}^{u} c_i(p) \cdot p^i$ for $p \in \mathbb{Z}$.

**Entire quasi-polynomials:** $EQP = \{ f \mid \forall p \in \mathbb{Z} : f(p) \in \mathbb{Z} \}$

**Example:**

$$f = \left[ \frac{3}{2}, \frac{1}{2} \right] X + \left[ 1, \frac{1}{2} \right] \in EQP$$

because $f(1) = \frac{1}{2} \cdot 1 + \frac{1}{2} = 1$, $f(2) = \frac{3}{2} \cdot 2 + 1 = 4$, etc.
Division with Remainder in $EQP$

- GCDs can be computed using division with remainder.
- We can define a kind of division with remainder in $EQP$, e.g.:
  \[
  X^2 = \left( \frac{1}{2}X - [0, \frac{1}{2}] \right) \cdot 2X + [0, 1]X
  \]
- Only complication: zero-divisors. No divisions in components that are zero.
This division in $\text{EQP}$ allows to construct finite remainder sequences:

\begin{align*}
    f_0 &= q_0 \cdot f_1 + f_2 & f_0(p) &= q_0(p) \cdot f_1(p) + f_2(p) \\
    f_1 &= q_1 \cdot f_2 + f_3 & f_1(p) &= q_1(p) \cdot f_2(p) + f_3(p) \\
    &\vdots & & \vdots \\
    f_{n-1} &= q_{n-1} \cdot f_n & f_{n-1}(p) &= q_{n-1}(p) \cdot f_n(p) \\
    \downarrow & & \downarrow \\
    f_n &= \gcd_{\text{EQP}}(f_0, f_1) & f_n(p) &= \gcd_{\mathbb{Z}}(f_0(p), f_1(p))
\end{align*}

$\gcd_{\text{EQP}}(f_0, f_1)(p) = \gcd_{\mathbb{Z}}(f_0(p), f_1(p))$
Weak and Pointwise Echelon Form

\[ S_1 = \begin{pmatrix} [1,0]X & 1 \\ 0 & 1 \end{pmatrix} \]

is in echelon form, because \([1,0]X \neq 0\) and \(1 \neq 0\).

But \(S_1(p)\) is not echelon for \(p = 0\), \(p \equiv_2 1\): \(S_1(p) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}\)

Serious problem: periodically vanishing pivots

Solution:

Additional row operations in the vanishing components.

\[ S_1 \rightsquigarrow S_2 = \begin{pmatrix} [1,0]X & 1 \\ 0 & [1,0] \end{pmatrix} \]

subtract first row times \([0,1]\) from second row

\(S_2(p)\) is echelon for all \(p \in \mathbb{Z} - M\), \(M = \{0\}\).
Entire quasi-polynomials allow to compute pointwise solutions of a system of linear Diophantine equations with one non-linear parameter.

This also generalises Banerjee's data dependence to one non-linear parameter.

Previously, only syntactic treatment of non-linearities (Pugh et al. 1995) or approximations.
Part 2:
Non-linearities in Transformations
Non-linear Transformations

Transformations may introduce non-linearities for different reasons, e.g.:

- Explicit non-linear schedules which are better than the best linear schedules (Achtziger et al. 2000),
- Non-linear parameter models a compile-time unknown (e.g. number of processors for tiling for a variable number of processors).
Some transformations (e.g., computing a schedule) can be expressed as quantifier elimination (QE) or QE with answer problems.

Unfortunately, QE is too slow even for small examples.

Alternative: Enhance a classical algorithm with the help of QE to handle non-linear parameters. Successful for, e.g.,
- Fourier-Motzkin elimination,
- Simplex,
- Chernikova's algorithm.

Classical Algorithm + QE

- Classical algorithms (like Simplex) make case distinctions on the signs of values in a coefficient matrix:

\[
\begin{pmatrix} 1 & 2 & -4 & 0 \end{pmatrix}
\]

\[
\begin{pmatrix} p & p^2 - q & -p & 0 \end{pmatrix}
\]

\[
\begin{align*}
\text{if } c & \geq 0 \text{ then } A \\
\text{else } & B
\end{align*}
\]

- With non-linear parameters, values are symbolic expressions in the parameters. → Case distinctions in the result.
- QE is used to prune paths with inconsistent conditions.
- Correctness by construction.
- Termination has to be proved.
Scheduling Example

Dependence:
\[ i \rightarrow i + n \]  
\[ n = 3 \]

Desired schedule:
\[ \theta(i) = \left\lfloor \frac{i}{n} \right\rfloor \]

Observations:
- Both QE with answer and Simplex+QE compute the desired schedule in a short time. (about 2 seconds on Core2Duo 2.4 GHz)
- QE with answer fails (is too slow or uses too much memory) for more complex examples (2-dimensional iteration domain, 2 dependences).
Tiling

- The parallelism often has to be coarsened by grouping operations into bigger chunks.
- Example: tiles with width $w$ and height $h$; Coordinates of the tiles: $(T,P)$

Non-linear transformations are becoming more desirable as we try to apply the polyhedron model to a wider range of programs or hardware.

Even “harmless” transformations may cause non-linearities to appear.
Part 3: Code Generation for Non-linearly Bounded Iteration Domains

```plaintext
for (t=1; t<=n; t++)
  parfor (p=1; p<=n-(n-t)^2; p++)
  ...
```
Non-linear Code Generation?

- Why non-linear code generation?
  - Non-linear parameters and variables are introduced by transformations (cf. Part 2).
  - A single non-linearity makes it impossible to use current code generation techniques (e.g., Bastoul 2004).

for \((x=a; x\leq d; x++)\) {
    for \((y=e; y\leq h; y++)\) {
        if \((a\leq x\leq c \land e\leq y\leq g)\) \(T_1\);
        if \((b\leq x\leq d \land f\leq y\leq h)\) \(T_2\);
    }
} 

For efficiency: 
No case distinctions inside the loops!
Compute partitionings of the iteration domains and their projections onto outer dimensions by
- intersections and differences of polyhedra,
- projections of polyhedra.

Invariant: intersections, differences and projections yield finite unions of polyhedra.
→ finitely many convex sets

Partitions (polyhedra) can be ordered in each dimension. The choice of the partitioning only affects the efficiency of the generated code.
Loops for Polyhedra with Non-linear Parameters

- Using QE we can generalise polyhedral code generation to non-linear parameters:
  - Fourier-Motzkin (or Chernikova) used to compute projections.
  - QE used to compute disjoint unions of polyhedra and ordering of polyhedra.
- The prototype implementation can generate code for all examples in CLooG's test suite.
Loops for Semi-algebraic Iteration Domains

- Semi-algebraic set = defined by polynomial (in-)equalities
- Can be non-convex:
  
  
  \begin{align*}
  1 & \leq x \leq 7 \\
  1 & \leq y \leq 9 \\
  0 & \leq (y - 4)^2 + 12 - 3x \\
  \end{align*}

- Convexity is not necessary for code generation.
- The analogy to dimension-wise ordered convex sets is **cylindrical** (algebraic) **decomposition**.
A Semi-algebraic Example

for (x=1; x≤4; x++)
  for (y=1; y≤9; y++)
    T(x,y);
for (x=4+1; x≤7; x++) {
  for (y=1; y≤\lfloor 4-\sqrt{3x-12} \rfloor; y++)
    T(x,y);
  for (y=\lceil 4+\sqrt{3x-12} \rceil; y≤9; y++)
    T(x,y);
}

1 ≤ x ≤ 7
1 ≤ y ≤ 9
0 ≤ (y - 4)^2 + 12 - 3x
Cylindrical Decomposition

Let \( R \subseteq \mathbb{R}^n \) connected, \( R \neq \emptyset \).

Then \( R \times \mathbb{R} \) is called a \textit{cylinder} over \( R \).

Let \( f_1, \ldots, f_r : R \to \mathbb{R} \) continuous and \( \forall x \in R : f_1(x) < f_2(x) < \cdots < f_r(x) \).

Then \((f_1, \ldots, f_r)\) defines a \textit{stack} over \( R \).

The graphs of the \( f_i \) are called \textit{sections}, and the regions between the graphs are called a \textit{sectors}.

Cylindrical \textit{algebraic} decomposition: \( f_i \) are roots of (multi-variate) polynomials.
Code for the Example

```cpp
for (x=1; x≤1; x++) {
    for (y=1; y≤1; y++)
        T(x,y);
    for (y=1+1; y≤9-1; y++)
        T(x,y);
    for (y=9; y≤9; y++)
        T(x,y);
}
for (x=1+1; x≤4-1; x++) {
    for (y=1; y≤1; y++)
        T(x,y);
    for (y=1+1; y≤9-1; y++)
        T(x,y);
    for (y=9; y≤9; y++)
        T(x,y);
}
for (x=4; x≤4; x++) {
    for (y=1; y≤1; y++)
        T(x,y);
    for (y=1+1; y≤4-1; y++)
        T(x,y);
    for (y=4; y≤4; y++)
        T(x,y);
    for (y=4+1; y≤9-1; y++)
        T(x,y);
    for (y=9; y≤9; y++)
        T(x,y);
}
for (x=4+1; x≤7-1; x++) {
    for (y=1; y≤1; y++)
        T(x,y);
    for (y=1+1; y≤[4−√3x−12]−1; y++)
        T(x,y);
    for (y=4−[3x−12]; y≤[4−√3x−12]; y++)
        T(x,y);
    for (y=4+√3x−12; y≤[4+√3x−12]; y++)
        T(x,y);
    for (y=4+√3x−12 +1; y≤9-1; y+)
        T(x,y);
    for (y=9; y≤9; y+)
        T(x,y);
}
for (x=7; x≤7; x++) {
    for (y=1; y≤1; y+)
        T(x,y);
    for (y=4+√3x−12; y≤[4+√3x−12]; y+)
        T(x,y);
    for (y=4+√3x−12 +1; y≤9-1; y+)
        T(x,y);
    for (y=9; y≤9; y+)
        T(x,y);
}
```
Generated code can be simplified automatically.
Code Generation Summary

- QE allows to generalise polyhedral code generation to non-linear parameters.
- Cylindrical decomposition enables to generate code for arbitrary semi-algebraic iteration domains.
- Prototypical implementations available:
  - Using FM/Chernikova+QE: NLGen (to be released soon). Can generate code for all of CLooG's test cases.
- Open question: relation of code generation to formula simplification (e.g., GEOFORM formulas)?
Conclusions

- The applicability of automatic loop parallelisation is restricted by many cases that are “slightly” outside the polyhedron model.
- In all three phases of the parallelisation process non-linearities can be handled.
- Dependence analysis is most challenging.
- Code generation is solved in theory.
- Quantifier elimination with answer is often too general and, therefore, too slow.
- Combining polyhedral methods (for polyhedral sub-problems) with the more general ones may improve the efficiency.