Representing Non-Affine Parallel Algorithms by means of Recursive Polyhedral Equations

Patrice Quinton$^1$ and Tomofumi Yuki$^2$

$^1$ENS Rennes
$^2$INRIA Rennes

Impact 2021, January 20, 2021
1 Introduction

2 Example: recursive minimum

3 Recursive ALPHA

4 Scheduling recursive ALPHA programs

5 Discussion

6 Conclusion
Outline

1. Introduction
2. Example: recursive minimum
3. Recursive ALPHA
4. Scheduling recursive ALPHA programs
5. Discussion
6. Conclusion
Introduction

- Polyhedral model: a powerful representation of computations for parallelism expression and extraction
- Limited by the expressivity of affine recurrence equations
- Extensions of the model have been proposed
- Divide-and-conquer programs difficult to represent, in a direct fashion
- Typical (and famous) limitation: FFT cannot be described
Content of this (on going) work

- Express divide-and-conquer algorithms using a polyhedral equational language
- Context: the ALPHA language
- How: extending the language with conditions on size parameter values
- What: show how affine scheduling can be extended by means of solving recursive equations to compute the parameter dependent part
- Basic idea: try to "confine" the problems to the parameter side.
Outline

1. Introduction

2. Example: recursive minimum

3. Recursive ALPHA

4. Scheduling recursive ALPHA programs

5. Discussion

6. Conclusion
Example: recursive minimum

Divide-and-conquer algorithm for computing the minimum of $N$ numbers

\[
\text{minRec}(A[N]) = \{
    \text{if } (N==1) \text{ return } A[0]; \\
    \text{left} = \text{minRec}(A[0:N/2]); \\
    \text{right} = \text{minRec}(A[N/2:N]); \\
    \text{return } \text{min(left, right)};
\}
\]
Call structure of recursive min when $N = 8$
Example: recursive minimum

**Alpha** sequential program to compute the minimum of \( N \) numbers

\[
\text{affine minValue}[N] \rightarrow \{ : 1 \leq N \} \\
in \\
array : \{ [i] : 1 \leq i \leq N \}; \\
out \\
minimum : \{ \}; \\
local \\
X : \{ [i] : 0 \leq i \leq N \}; \\
let \\
X[i] = \text{case} \{ \\
\{ : i = 0 \} : 0[]; \\
\{ : 0 < i \} : \text{min} (X[i - 1], \text{array}[i]); \\
\}; \\
minimum = X[N]; \\
\]

(1)
Syntax of ALPHA

- Parameters: \([N] \rightarrow \{ : 1 \leq N\}\)
- Domains: array : \({[i] : 1 \leq i \leq N}\)
- Equations:

\[
X[i] = \begin{cases} 
: i=0 & : 0[ ] ; \\
: 0 < i & : \text{min} ( X[i-1], \text{array}[i] ) ; 
\end{cases}
\]

Remark: ALPHA expressions are **functional**, allowing formal transformations to be clearly defined (See [Mauras, 1989])

Outline

1. Introduction
2. Example: recursive minimum
3. Recursive ALPHA
4. Scheduling recursive ALPHA programs
5. Discussion
6. Conclusion
Recursive \textbf{Alpha}

\section*{Calls to subsystems}

\[(Y_1, Y_2, \ldots, Y_m) = <name>[f](X_1, X_2, \ldots, X_n)\]

- \textit{name} is the name of the subsystem called
- \(X_i\) are the inputs
- \(Y_i\) are the outputs
- \(f\) is an affine function of the parameters. \(f = (p \rightarrow f(p) = q)\)

\section*{When clauses}

A \textbf{when} clause governs a set of equations (or system calls) that apply when some condition on the parameter is met
Recursive minRec (1/2)

Base case

```
affine minRec[N] -> { 1 <= N }

in

array: {[i] : 1 <= i <= N}

out

minimum: {}

when { N = 1 }

let

minimum = array[1];
```
Recursive minRec (2/2)

Recursive part

\[
\text{when } \{ : N \geq 2 \} \\
\text{local} \\
\min1 : \{\} \\
\min2 : \{\} \\
\text{array1} : \{[i] : 1 \leq i \text{ and } 2i \leq N\} \\
\text{array2} : \{[i] : 1 \leq i \text{ and } 2i \leq N\} \\
\text{let} \\
\text{array1}[i] = \text{array}[i]; \\
\text{array2}[i] = \text{array}[i + N/2]; \\
(\min1) = \text{minRec}[N/2](\text{array1}); \\
(\min2) = \text{minRec}[N/2](\text{array2}); \\
\text{minimum} = \text{min} (\min1, \min2); 
\]
Outline

1. Introduction
2. Example: recursive minimum
3. Recursive ALPHA
4. Scheduling recursive ALPHA programs
5. Discussion
6. Conclusion
Scheduling standard \textbf{ALPHA} (1/3)

- \( V \), \textit{variable}, \( V(z) \) \textit{value} of \( V \) at point \( z \)
- \( t_V(z) \) denotes the \textit{schedule} of \( V \)
- \( t_V(z) > t_W(z') \) whenever \( V(z) \) \textit{depends} on \( W(z') \)
- In a \textit{standard ALPHA program} with parameter \( p \),
  \[ t_V(z) = \tau_V.z + \alpha_V + \sigma_V.p \]
- Schedule found by enforcing causality in \textit{each point} of the \textit{domain} of \( V \) using either :
  - the \textit{Farkas} method (Feautrier)
  - the \textit{vertex} method (Quinton et al.)
- In both cases, ILP of a few tens of inequalities
Scheduling standard **ALPHA**: calls to subsystems (2/3)

\[
\begin{align*}
n &= 0 \\
X &= t = a_i + b \\
\text{SYS} &= t = a_i + b + d \\
Y &= t = a_i + b + e \\
\text{Call 1 to SYS} &= \text{Call 2 to SYS} \\
Z &= t = e \\
\end{align*}
\]
To schedule systems including subsystem calls:

- Assume subsystem is already scheduled
- Gather schedule of inputs and outputs and add the same unknown expression (possibly depending on the parameter) to the schedule of the call
- Enforce the dependencies between I/O of system and their actual value in the calling system
- Remark: other, more sophisticated, methods exist.
Scheduling recursive **ALPHA**

**Assumption and remarks**

- **Simple recursion** scheme (no mutually recursive systems)
- Schedule function affine, **uni-dimensional**, except parameter term
- **Cannot assume** that subsystem is scheduled

**Method**

- Assume that schedule has the form $t_V(z) = \tau_V z + \alpha_V + \phi(p)$ where $\phi$ is a function **to be determined**.
- For equations, proceed **as in the standard case**
- For system calls, take into account the parameter mapping function $f$, and **separate** the computation of the $\tau_V, \alpha_V$ and of $\phi$
Recursion equations

Let $P_b$ be the parameter domain of the base part, and $P_r$ that of the recursive part.

$$\phi(p) = \begin{cases} 
p_0 & \text{if } p \in P_b \\
\phi(f(p)) + 1 & \text{if } p \in P_r 
\end{cases} \quad (2)$$

Example of minRec

$$\phi(p) = \begin{cases} 
1 & \text{if } p = 1 \\
p/2 + 1 & \text{if } p > 1 
\end{cases} \quad (3)$$

Solution: $\phi(p) = \log_2 p + 1$

For more general cases, see [Cormen et al., 2001] or [Benoît et al., 2013]
Outline

1. Introduction
2. Example: recursive minimum
3. Recursive ALPHA
4. Scheduling recursive ALPHA programs
5. Discussion
6. Conclusion
**Discussion**

**FFT (1/2)**

```plaintext
affine FFT[N] -> {1 ≤ N}

in
  x : {[i] : 1 ≤ i ≤ N}

out
  y : {[i] : 1 ≤ i ≤ N}

when {N = 1}

let
  y[i] = x;

when {2 ≤ N}

local
  left : {[i] : 1 ≤ i and 2i ≤ N}
  right : {[i] : 1 ≤ i and 2i ≤ N}
  q1 : {[i] : 1 ≤ i and 2i ≤ N}
  q2 : {[i] : 1 ≤ i and 2i ≤ N}
  z : {[i] : 1 ≤ i ≤ N}.
```

P. Quinton and T. Yuki

Recursive Polyhedral Equations

Impact 2021 22 / 30
let
left[i] = x[2*i - 1];  -- Separate left an right
right[i] = x[2*i];
(q1) = FFT[N/2](left);  -- Recursive call
(q2) = FFT[N/2](right);
-- Sketch of butterfly computation
z[i] =
    case {
        { : 2i ≤ N} : if i%2 = 0 then q1[i] + q1[i - 1]
                    else q1[i] + q1[1 + i];
        { : N < 2i} : if i%2 = 0 then q2[i - N/2] +
                       q2[1 + i - N/2]
                    else q2[i - N/2] + q2[1 + i - N/2];
    };
    -- Set result
    y[i] = z[i];
.
Discussion

- FFT can be represented and scheduled (done using MMAAlpha)
- Divide-and-conquer with other ratios can be easily covered
- Static analysis allows checking that program follows assumptions
- Theoretical complexity is that of ILP, but in practice, not a problem
- Open: multi-dimensional schedule, mutually recursive programs
- References to other approaches of polyhedral recursion in the paper
Outline

1. Introduction
2. Example: recursive minimum
3. Recursive ALPHA
4. Scheduling recursive ALPHA programs
5. Discussion
6. Conclusion
Conclusion

Summary

- Divide-and-conquer algorithms modelization for the polyhedral model
- Structured scheduling of affine equations extended to recursive program
- Representation and parallelization of FFT can be done
- Basic properties of polyhedral equations are preserved

Future work

- Implement change of basis, etc.
- Extend to multi-dimensional scheduling
- Implement VHDL code generation
- Combine recursivity and reductions for high-level transformations
Thank you!
Experiments and Numbers

- MMAlpha: implementation of Alpha workflow based on Mathematica, using the Polyhedral Library
- Scheduling based on the vertex method, using the ILP solver of Mathematica (Interior point method)
- Typical scheduling time: 48 equations, 1066 inequalities, 1.61 s (MacBook Pro, 2.3GHz)
- FFT (recursive) scheduling:
  - Finding out the \( \tau \)'s and \( \alpha \)'s: 0.18 s
  - Solving the recursions: 0.18 s
Other works

- **Extensions** of the Polyhedral Model [Benabderrahamne et al, 2010], [Ioss et al., 2014]
- Use *dynamic compilation* to discover hidden polyhedral parts [Kobeissi and Clauss, 2019]
- **Transformation** of recursive programs [Sudararajah and Kulkarni, 2015]
- **Space exploration** through linear transforms (SPIRAL) [Franchetti at al, 2018]
- Divide-and-conquer for *dynamic programming* [Javanmard et al, 2020]
The two branches of the Polyhedral Model

A little bit of archeology

- Loop parallelization [Kuck, circa 1970]
- Modelization by recurrence equations [Karp et al., circa 1970]
- Systolic array modelization [Moldovan, Quinton, circa 1980]
- Data-flow analysis [Feautrier, 1991]
- Alpha language [Mauras, 1989]

Current situation

- Branch 1: analysis of loops, dependence analysis, loop rewriting
- Branch 2: expression of computations, program transformations
- Sharing many methods and techniques