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

Outline

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

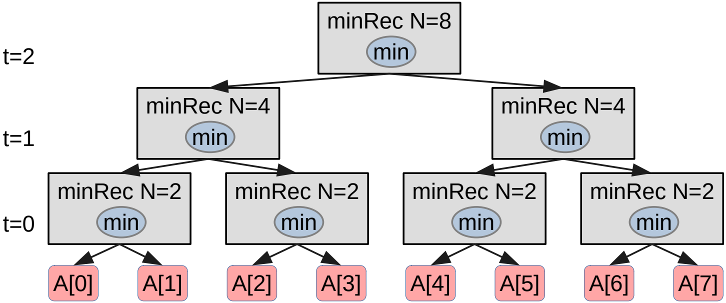
- Express `divide-and-conquer` algorithms using a `polyhedral equational language`
- `Context` : the ALPHA language
- How : extending the language with `conditions on size parameter values`
- What : show how affine scheduling can be extended by means of `solving recursive equations` to compute the parameter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

Outline

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

```
minRec(A[N]) = {
    if (N==1) return A[0];
    left = minRec(A[0:N/2]);
    right = minRec(A[N/2:N]);
    return min(left, right);
}
```

Call structure of recursive min when $N = 8$



ALPHA sequential program to compute the minimum of N numbers

$$
\begin{array}{l}
\textbf{affine}\ \mathsf{minValue}[N] \rightarrow \{: 1 \leq N\} \\
\textbf{in} \\
\quad \mathsf{array} : \{[i] : 1 \leq i \leq N\}; \\
\textbf{out} \\
\quad \mathsf{minimum} : \{\}; \\
\textbf{local} \\
\quad \mathsf{X} : \{[i] : 0 \leq i \leq N\}; \\
\textbf{let} \\
\quad \mathsf{X}[i] = \textbf{case}\ \{ \\
\qquad \{: i = 0\} : 0[]; \\
\qquad \{: 0 < i\} :\ \textbf{min}\ (\mathsf{X}[i-1], \mathsf{array}[i]); \\
\quad \}; \\
\quad \mathsf{minimum} = \mathsf{X}[N]; \\
.
\end{array}
\tag{1}
$$

Syntax of Alpha

- Parameters: $[N] \to \{: 1 \leq N\}$
- Domains: array : $\{[i] : 1 \leq i \leq N\}$
- Equations:

```
X[i] =
   case
      { :i=0} : 0[] ;
      { :0 < i} :  min  ( X[i-1], array[i] ) ;
   esac ;
```

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

Outline

Recursive ALPHA

Calls to subsystems

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

- $name$ 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. $f = (p \to f(p) = q)$

When clauses

A **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

$$
\begin{aligned}
&\textbf{affine}\ \text{minRec}[N] \rightarrow \{: 1 \leq N\} \\
&\textbf{in} \\
&\quad \text{array} : \{[i] : 1 \leq i \leq N\} \\
&\textbf{out} \\
&\quad \text{minimum} : \{\} \\
&\textbf{when}\ \{: N = 1\} \\
&\textbf{let} \\
&\quad \text{minimum} = \text{array}[1]; \\
&.
\end{aligned}
$$

Recursive `minRec` (2/2)

Recursive part

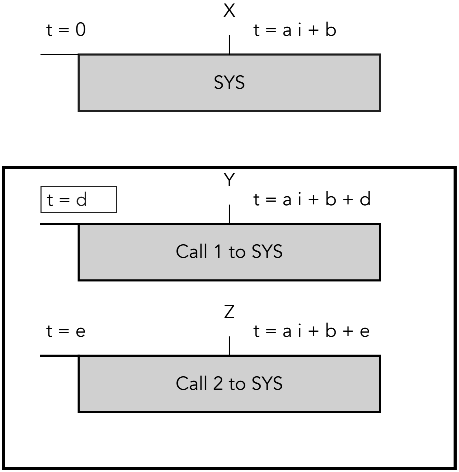
when $\{: N \geq 2\}$
local
  min1 : $\{\}$
  min2 : $\{\}$
  array1 : $\{[i] : 1 \leq i \textbf{ and } 2i \leq N\}$
  array2 : $\{[i] : 1 \leq i \textbf{ and } 2i \leq N\}$
let
  array1$[i]$ = array$[i]$;
  array2$[i]$ = array$[i + N/2]$;
  (min1) = minRec$[N/2]$(array1);
  (min2) = minRec$[N/2]$(array2);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

Outline

Scheduling standard ALPHA (1/3)

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

Scheduling standard ALPHA : calls to subsystems (2/3)



Scheduling standard ALPHA (2/3)

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 ϕ 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 τ_V, α_V and of ϕ

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

FFT (1/2)

affine FFT$[N] \rightarrow \{: 1 \leq N\}$
in
    x : $\{[i] : 1 \leq i \leq N\}$
out
    y : $\{[i] : 1 \leq i \leq N\}$
when $\{: N = 1\}$
let
    y$[i]$ = x;
.
when $\{: 2 \leq N\}$
local
    left : $\{[i] : 1 \leq i \textbf{ and } 2i \leq N\}$
    right : $\{[i] : 1 \leq i \textbf{ and } 2i \leq N\}$
    q1 : $\{[i] : 1 \leq i \textbf{ and } 2i \leq N\}$
    q2 : $\{[i] : 1 \leq i \textbf{ and } 2i \leq N\}$
    z : $\{[i] : 1 \leq i \leq N\}$.

FFT (2/2)

let

$\text{left}[i] = \text{x}[2 * i - 1];$ -- Separate left an right
$\text{right}[i] = \text{x}[2 * i];$
$(\text{q1}) = \text{FFT}[N/2](\text{left});$ -- Recursive call
$(\text{q2}) = \text{FFT}[N/2](\text{right});$
-- Sketch of butterfly computation
$\text{z}[i] =$
case $\{$
$\{: 2i \leq N\}$: **if** $i\%2 = 0$ **then** $\text{q1}[i] + \text{q1}[i-1]$
else $\text{q1}[i] + \text{q1}[1+i];$
$\{: N < 2i\}$: **if** $i\%2 = 0$ **then** $\text{q2}[i - N/2] +$
$\text{q2}[1 + i - N/2]$
else $\text{q2}[i - N/2] + \text{q2}[1 + i - N/2];$
$\};$
-- Set result
$\text{y}[i] = \text{z}[i];$

.

Discussion

- FFT can be `represented and scheduled` (done using MMAlpha)
- 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

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 τ's and α'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