Estimation of Parallel Complexity with Rewriting Techniques

Christophe Alias[*], Carsten Fuhs[†], Laure Gonnord[‡]

[*] INRIA & LIP (UMR CNRS/ENS Lyon/UCB Lyon1/INRIA), Lyon, France,
christophe.alias@ens-lyon.fr
[†] Birkbeck, University of London, United Kingdom,
carsten@dcs.bbk.ac.uk
[‡] University of Lyon & LIP (UMR CNRS/ENS Lyon/UCB Lyon1/INRIA), Lyon,
France, laure.gonnord@ens-lyon.fr

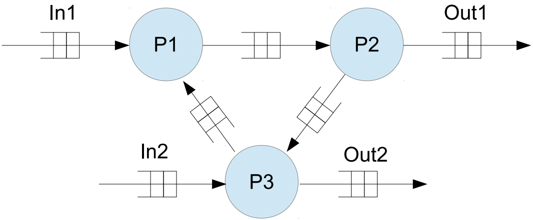
Languages, Compilation and Semantics, LIP Seminar
November, 3rd 2016

Outline

1. Introduction
2. Parallel complexity of regular programs
   Parallelization $\mapsto$ Termination
3. Parallel complexity of recursive programs
   Termination $\mapsto$ Parallelization
4. Conclusions

Context: Verification of Parallel Systems...

Parallel Systems



$\rightsquigarrow$ Many models: KPN (and variants), SDF (and variants), etc

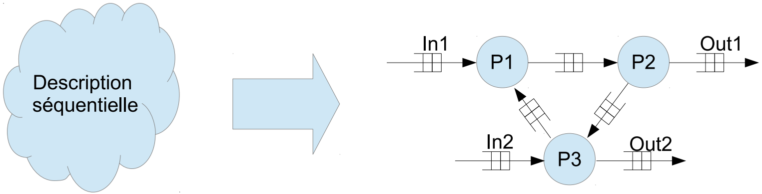
Correctness

- Determinism
- Termination

Efficiency

- Latency
- Bandwidth

… generated by parallelizing compilers!



From the source code:

- Target latency? $\geq$ termination
- Target bandwidth?

Inner Parallelism, I

Trace $(\Omega_{\mathcal{I}}, \prec_{\mathsf{seq}})$: sequence of operations executed by the program on the input $\mathcal{I}$.

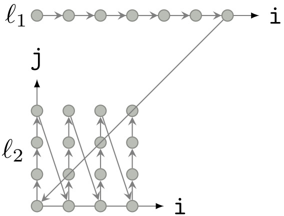
Undecidable in general!

- Restrict program model $\rightsquigarrow$ Polyhedral model
- Over-approximate $\Omega_{\mathcal{I}}$ $\rightsquigarrow$ Abstract interpretation

Regular Program $\Omega_{\mathcal{I}}$ does not depend on $\mathcal{I}$.
$\rightsquigarrow$ Polyhedral model

Polyhedral Model: Affine Loop Nests

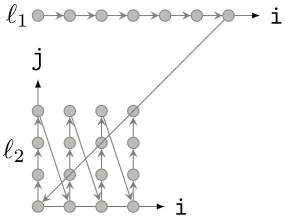
for i := 0 to 2*N
  ℓ_1: c[i] := 0;
for i := 0 to N
  for j := 0 to N
    ℓ_2: c[i+j] := c[i+j] + a[i]*b[j];



- for loop with arrays + affine constraints

Polyhedral Model: Affine Loop Nests

```
for i := 0 to 2*N
  ℓ1: c[i] := 0;
for i := 0 to N
  for j := 0 to N
    ℓ2: c[i+j] := c[i+j] + a[i]*b[j];
```



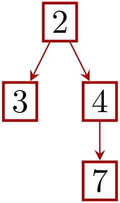
- `for` loop with arrays + affine constraints
- $\Omega_{\mathcal{I}}$ can be encoded with integer polyhedra:

$$\begin{array}{rcl} \langle \ell_1, i \rangle & : & i \in [\![0, 2N]\!] \\ \langle \ell_2, i, j \rangle & : & (i, j) \in [\![0, N]\!]^2 \end{array}$$

- Static analysis with ILP: dependences, scheduling, allocation

Tree Model: Tree Traversals

```java
public int treeMax() {
  int leftMax = Integer.MIN_VALUE;
  int rightMax = Integer.MIN_VALUE;
  if (this.left != null)
    leftMax = this.left.treeMax();
  if (this.right != null)
    rightMax = this.right.treeMax();
  return Math.max(this.val,
            Math.max(leftMax, rightMax));
}
```



- Each node (subtree) of t is an operation of Ω_t
- Not exactly regular, but Ω_t is decidable!

Data Dependence relate communicating/conficting operations,

$\rightarrow \subseteq \Omega \times \Omega$

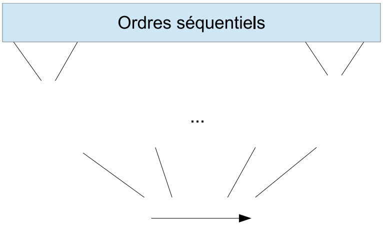
$\rightsquigarrow$ usually split into flow-, anti-, output-dependences

Schedule Assign dates to operations, $\theta : \Omega \rightarrow (\mathcal{D}, \prec)$

Induced order $\prec_\theta = \{(s,t),\ \theta(s) \prec \theta(t)\}$

Correctness $s \rightarrow t \implies \theta(s) \prec \theta(t)$

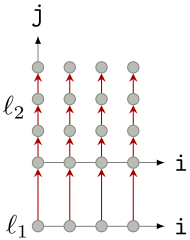
Hence: $\rightarrow \subseteq \prec_\theta$



Latency

```
//Compute y = Ax
for i := 0 to N-1
  ℓ1: y[i] := 0;
  for j := 0 to N-1
    ℓ2: y[i] := y[i] + a[i][j]*x[j];
```

$\boxed{\lambda = \mathcal{O}(N)}$



Latency λ longest chain of $\rightarrow$

Degree of sequentiality smallest $s \in \mathbb{N}$ s.t. $\lambda = \mathcal{O}(N^s)$

Counterpart in the Tree Model?

Links with Termination Analysis

Transition system

- Σ set of states
- $\Sigma_0 \subseteq \Sigma$ initial states
- $\rightarrow \subseteq \Sigma \times \Sigma$ transition relation

Termination Exhibit a ranking function $\rho : \Sigma \rightarrow (\mathcal{D}, \prec)$ s.t.

$$s \rightarrow t \Rightarrow \rho(t) \prec \rho(s)$$

WCET

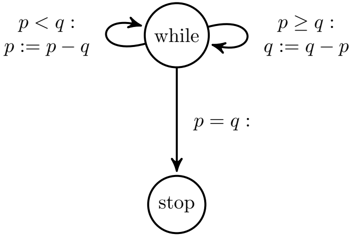
$$\lambda = \max\{n,\ \exists \sigma_0 \in \Sigma_0 : \sigma_0 \rightarrow^n \sigma\}$$

Upper bound:

$$\lambda \leq \#\{\rho(\sigma),\ \exists \sigma_0 \in \Sigma_0 : \sigma_0 \rightarrow^* \sigma\}$$

Instance: Integer Transition Systems

```
while(p ≠ q)
  if(p<q)
    p := p - q;
  else
    q := q - p;
```



- Each state of Σ is a pair $\langle \ell, \vec{i} \rangle$
- ρ is affine per control point:

$$\rho : \langle \ell, \vec{i} \rangle \mapsto Ai + b, \qquad \Sigma \to (\mathbb{Z}^p, \ll)$$

Cross Fertilization

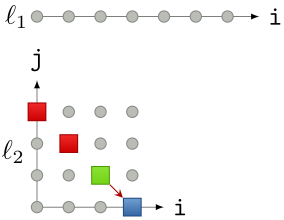
Parallelization	Termination
Dependence	Transition system
Schedule	Ranking function
Latency	WCET

Outline

1. Introduction
2. Parallel complexity of regular programs
   Parallelization $\mapsto$ Termination
3. Parallel complexity of recursive programs
   Termination $\mapsto$ Parallelization
4. Conclusions

Dependence Analysis

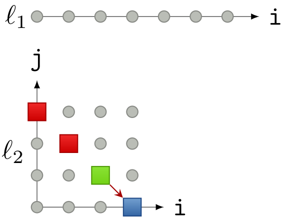
```
for i := 0 to 2*N
  ℓ1: c[i] := 0;
for i := 0 to N
  for j := 0 to N
    ℓ2: c[i+j] := c[i+j] + a[i]*b[j];
```



- Idea: Given a consumer, find the last producer $\rightsquigarrow$ ILP.

Dependence Analysis

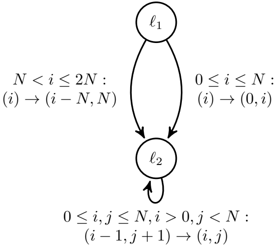
```
for i := 0 to 2*N
  ℓ1: c[i] := 0;
for i := 0 to N
  for j := 0 to N
    ℓ2: c[i+j] := c[i+j] + a[i]*b[j];
```

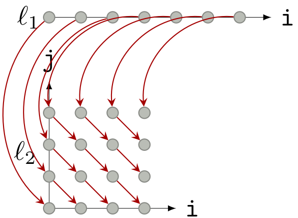


- Idea: Given a consumer, find the last producer $\rightsquigarrow$ ILP.
- $\to_N$ is an affine relation:

$$
\begin{array}{rll}
\langle \ell_2, i-1, j+1 \rangle & \to_N \langle \ell_2, i, j \rangle & : i > 0 \wedge j < N \\
\langle \ell_1, i \rangle & \to_N \langle \ell_2, 0, i \rangle & : 0 \leq i \leq N \\
\langle \ell_1, i \rangle & \to_N \langle \ell_2, i-N, N \rangle & : N < i \leq 2N
\end{array}
$$

Dependence Analysis – Representation



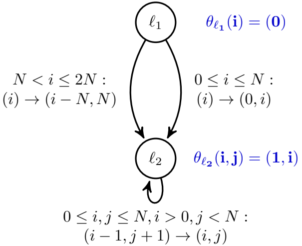


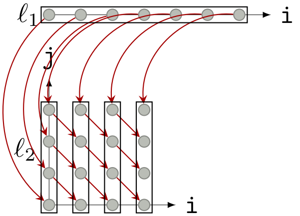
- $\to_N$ is an affine relation:

$$
\begin{array}{rcll}
\langle \ell_2, i-1, j+1 \rangle & \to_N & \langle \ell_2, i, j \rangle & : i > 0 \wedge j < N \\
\langle \ell_1, i \rangle & \to_N & \langle \ell_2, 0, i \rangle & : 0 \leq i \leq N \\
\langle \ell_1, i \rangle & \to_N & \langle \ell_2, i - N, N \rangle & : N < i \leq 2N
\end{array}
$$

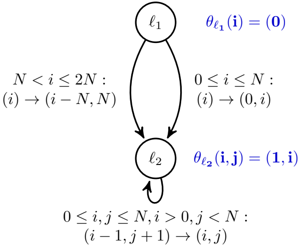
- ... finitely represented as a graph (PRDG)

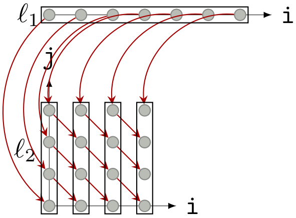
Scheduling





- Timestamps are vectors of $(\mathbb{N}^{d_\ell}, \ll)$.
- Affine schedule: $\theta_\ell : \vec{x} \mapsto A_\ell \vec{x} + B_\ell \vec{N} + \vec{c}_\ell \quad \mathcal{D}_\ell \to (\mathbb{N}^{d_\ell}, \ll)$
- Can be computed with ILP





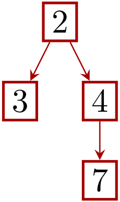
- Timestamps are vectors of $(\mathbb{N}^{d_\ell}, \ll)$.
- Affine schedule: $\theta_\ell : \vec{x} \mapsto A_\ell \vec{x} + B_\ell \vec{N} + \vec{c}_\ell \quad \mathcal{D}_\ell \rightarrow (\mathbb{N}^{d_\ell}, \ll)$
- Can be computed with ILP
- Bonus: reverse the order: termination algorithm! [RanK, 2010]

Outline

1. Introduction
2. Parallel complexity of regular programs
   Parallelization $\mapsto$ Termination
3. Parallel complexity of recursive programs
   Termination $\mapsto$ Parallelization
4. Conclusions

Target: recursive programs on trees

```
public int treeMax() {
  int leftMax = Integer.MIN_VALUE;
  int rightMax = Integer.MIN_VALUE;
  if (this.left != null)
    leftMax = this.left.treeMax();
  if (this.right != null)
    rightMax = this.right.treeMax();
  return Math.max(this.val,
            Math.max(leftMax, rightMax));
}
```



- Each node (subtree) of t is an operation of Ω_t.
- $\rightarrow$ can be encoded as a term rewrite system (TRS):

$$\begin{aligned} \mathrm{dep}(\mathrm{Tree}(\mathit{val}, \mathit{left}, \mathit{right})) &\rightarrow \mathrm{dep}(\mathit{left}) \\ \mathrm{dep}(\mathrm{Tree}(\mathit{val}, \mathit{left}, \mathit{right})) &\rightarrow \mathrm{dep}(\mathit{right}) \end{aligned}$$

- How to schedule (check the termination of) a TRS?

$\rightsquigarrow$ With monotone interpretations! [AProVE, KoAT]

Monotone Interpretations

Given a TRS $\rightarrow$ over a term algebra $\mathcal{T}(\Sigma)$:

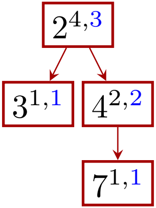
- Assign each symbol $a/n \in \Sigma$ with $[a] : \mathbf{R}^n \rightarrow \mathbf{R}$.
- The monotone interpretation of a term is:

$$[\![f(t_1, \ldots, t_n)]\!] = [f]([\![t_1]\!], \ldots, [\![t_n]\!])$$

- Correctness: $t \rightarrow u \Rightarrow [\![t]\!] > [\![u]\!]$.
- Polynomial interpretations can be computed by KoAT.

Putting it all together

```java
public int treeMax() {
  int leftMax = Integer.MIN_VALUE;
  int rightMax = Integer.MIN_VALUE;
  if (this.left != null)
    leftMax = this.left.treeMax();
  if (this.right != null)
    rightMax = this.right.treeMax();
  return Math.max(this.val,
          Math.max(leftMax, rightMax));
}
```



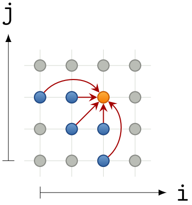
Monotone interpretation	Parallel complexity
$[\mathrm{dep}](x_1) = x_1$ $[\mathrm{Tree}](x_1, x_2, x_3) = x_2 + x_3 + 1$	$\lambda = \mathcal{O}(\mathrm{height}(t))$
$[\mathrm{dep}](x_1) = x_1$ $[\mathrm{Tree}](x_1, x_2, x_3) = \max(x_2, x_3) + 1$	$\lambda = \mathcal{O}(\mathrm{height}(t))$

What happens on polyhedral programs?

```
for(i=0; i<=N; i++)
  for (j=0; j<=N; j++)
    //Block S
    {
      m1[i][j] = Integer.MIN_VALUE;
      for(k=1; k<=i; k++)
        m1[i][j] = max(m1[i][j],H[i-k][j] + W[k]);

      m2[i][j] = Integer.MIN_VALUE;
      for(k=1; k<=j; k++)
        m2[i][j] = max(m2[i][j],H[i][j-k] + W[k]);

      H[i][j] = max(0,H(i-1,j-1)+s(a[i],b[i]),
                       m1[i][j],m2[i][j]);
    }
```



$$
\begin{aligned}
\mathrm{dep}(i,j) &\to \mathrm{dep}(i-1,j-1) : 0 \le i \le n, 0 \le j \le n \\
\mathrm{dep}(i,j) &\to \mathrm{dep}(i-k,j) : 0 \le i \le n, 0 \le j \le n, 1 \le k \le i \\
\mathrm{dep}(i,j) &\to \mathrm{dep}(i,j-\ell) : 0 \le i \le n, 0 \le j \le n, 1 \le \ell \le j
\end{aligned}
$$

Result: $[\mathrm{dep}](x_1, x_2) = x_1 + x_2 \qquad \lambda \le 2n$

Same as in the polyhedral model!

Conclusions

Position:

- Two successful experiences of cross fertilization *Parallelization* $\leftrightarrow$ *Termination*.
- Monotonic interpretations can benefit to automatic parallelization.
- Natural extension of affine scheduling to recursive programs

Locks:

- How to define/find the best schedule?
- How to count the steps?
- Steps towards a parallelizing compiler:
    - Computation partitioning?
    - Generation of the parallel code given a schedule?

Thanks!