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

15th International Workshop on Termination
September 5, 2016, Obergurgl, Austria

Why Automatic Parallelization?

- Most computers are parallel (end of Dennard scaling...)
- Writing/debugging a parallel program is (horribly) difficult

$\rightsquigarrow$ Automation is required...

Context: HPC's Automatic Parallelization

Why Automatic Parallelization?

- Most computers are parallel (end of Dennard scaling...)
- Writing/debugging a parallel program is (horribly) difficult

$\rightsquigarrow$ Automation is required...

Challenges:

- How to represent the computation? $\rightsquigarrow$ data dependencies
- How much parallelism? $\rightsquigarrow$ data dependencies
- Which parallelism (scheduling)? $\rightsquigarrow$ data dependencies
- Which resource allocation? $\rightsquigarrow$ data dependencies

Context: HPC's Automatic Parallelization

Why Automatic Parallelization?

- Most computers are parallel (end of Dennard scaling...)
- Writing/debugging a parallel program is (horribly) difficult

$\rightsquigarrow$ Automation is required...

Challenges:

- How to represent the computation? $\rightsquigarrow$ data dependencies
- How much parallelism? $\rightsquigarrow$ data dependencies
- Which parallelism (scheduling)? $\rightsquigarrow$ data dependencies
- Which resource allocation? $\rightsquigarrow$ data dependencies

☹ Bad news: checking data dependencies is undecidable.

Related Work and Contributions

Focus on regular imperative programs (polyhedral model)

- ☺ Unifying framework for program parallelization
- ☺ Exact set of dependencies, all the parallelism is found
- ☹ Scalability issues

Related Work and Contributions

Focus on regular imperative programs (polyhedral model)

- ☺ Unifying framework for program parallelization
- ☺ Exact set of dependencies, all the parallelism is found
- ☹ Scalability issues

Over-approximate the data dependencies

- ☺ Scalable
- ☹ Can be (very) rough and miss most of the parallelism

Related Work and Contributions

Focus on regular imperative programs (polyhedral model)

- ☺ Unifying framework for program parallelization
- ☺ Exact set of dependencies, all the parallelism is found
- ☹ Scalability issues

Over-approximate the data dependencies

- ☺ Scalable
- ☹ Can be (very) rough and miss most of the parallelism

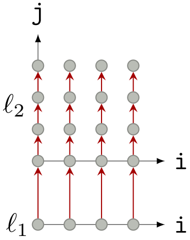
Contributions:

- Assess the parallel complexity of (some) recursive programs
- … using monotone interpretations
- Extends of the polyhedral model to recursive programs

Parallel Complexity

```
//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];
```

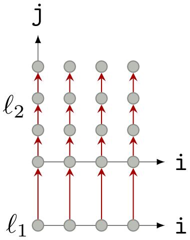
$\boxed{\lambda_N = \mathcal{O}(N)}$



- Minimum number of (parallel) computation steps assuming unbounded parallel resources.
- Solved on regular programs (polyhedral model)
- Goal: recursive programs on trees!

for i := 0 to N-1
  ℓ_1: y[i] := 0;
  for j := 0 to N-1
    ℓ_2: y[i] := y[i] + a[i][j]*x[j];

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

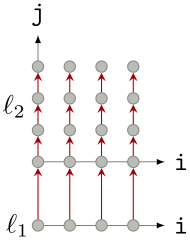


1. Divide an execution e into a sequence of operations $\mathcal{O}_e$
   $\rightarrow$ How to represent/approximate e? which grain?

Parallel Complexity – Methodology

```
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];
```

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

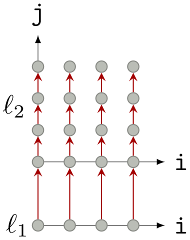


1. Divide an execution e into a sequence of operations $\mathcal{O}_e$
   $\rightarrow$ How to represent/approximate e? which grain?
2. Compute the dependencies: $\rightarrow_e \subseteq \mathcal{O}_e \times \mathcal{O}_e$
   $\rightarrow$ Impact of approximation?

Parallel Complexity – Methodology

for i := 0 to N-1
  ℓ_1: y[i] := 0;
  for j := 0 to N-1
    ℓ_2: y[i] := y[i] + a[i][j]*x[j];

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



1. Divide an execution e into a sequence of operations $\mathcal{O}_e$
   $\rightarrow$ How to represent/approximate e? which grain?
2. Compute the dependencies: $\rightarrow_e \subseteq \mathcal{O}_e \times \mathcal{O}_e$
   $\rightarrow$ Impact of approximation?
3. Compute the parallel complexity: $\lambda_e :=$ height$(\rightarrow_e)$
   $\rightarrow$ How to express λ_e?

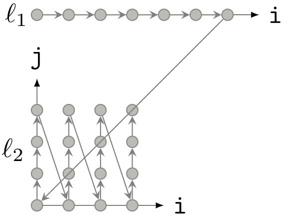
Outline

1. Parallel complexity of regular programs

2. Parallel complexity of recursive programs

Polyhedral Model at a Glance

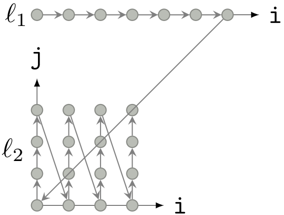
for i := 0 to 2*N
  ℓ_1: c[i] := 0;
for i := 0 to N
  for j := 0 to N
    ℓ_2: c[i+j] := c[i+j] + a[i]*b[j];



- Automatic parallelization of regular loop nests with arrays

Polyhedral Model at a Glance

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];



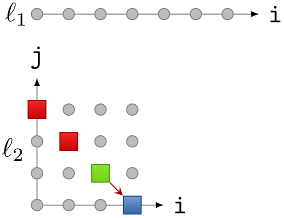
- Automatic parallelization of regular loop nests with arrays
- e is decidable and can be analyzed (*e.g.* with ILP)

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

- Key analysis: array dependencies, affine scheduling

(Affine) Array Dependencies

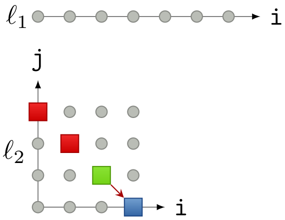
```
for i := 0 to 2*N
  ℓ1: c[i] := 0;
for i := 0 to N
  for j := 0 to N
    ℓ2: c[i+j] := c[i+j] + a[i]*b[j];
```



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

(Affine) Array Dependencies

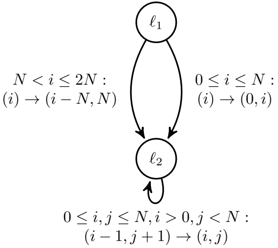
```
for i := 0 to 2*N
  ℓ1: c[i] := 0;
for i := 0 to N
  for j := 0 to N
    ℓ2: c[i+j] := c[i+j] + a[i]*b[j];
```

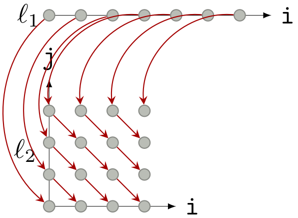


- Idea: Given a consumer, find the last producer $\rightsquigarrow$ ILP.
- $\to_N$ is an affine relation:

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

(Affine) Array Dependencies

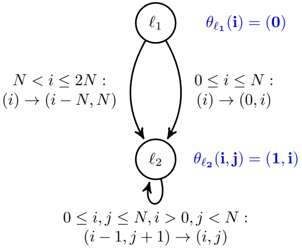


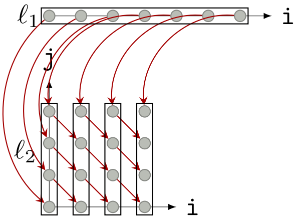


- Idea: Given a consumer, find the last producer $\rightsquigarrow$ ILP.
- $\to_N$ is an affine relation:

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

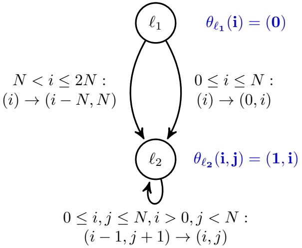
(Affine) Scheduling

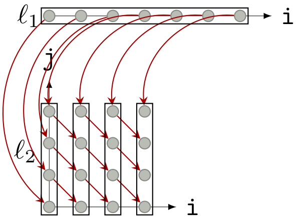




- Assign each operation $\langle \ell, \vec{x} \rangle$ with a timestamp $\theta_\ell(\vec{x}) \in \mathbb{N}^{d_\ell}$.
- Correctness: $\langle \ell, \vec{x} \rangle \to_N \langle \ell', \vec{y} \rangle \Rightarrow \theta_\ell(\vec{x}) \ll \theta_{\ell'}(\vec{y})$
- Affine schedule: $\theta_\ell(\vec{x}) = A\vec{x} + \vec{b} \rightsquigarrow$ ILP.

(Affine) Scheduling





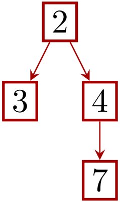
- Assign each operation $\langle \ell, \vec{x} \rangle$ with a timestamp $\theta_\ell(\vec{x}) \in \mathbb{N}^{d_\ell}$.
- Correctness: $\langle \ell, \vec{x} \rangle \to_N \langle \ell', \vec{y} \rangle \Rightarrow \theta_\ell(\vec{x}) \ll \theta_{\ell'}(\vec{y})$
- Affine schedule: $\theta_\ell(\vec{x}) = A\vec{x} + \vec{b} \rightsquigarrow$ ILP.
- Bonus: reverse the order: termination algorithm! [RanK, 2010]

Cross Fertilization

HPC community	TCS community
Data Dependence Graph Schedule Latency	Integer transition system Ranking function Computational complexity
Recursive schedule	Monotonic interpretations

Target: recursive programs on trees

```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 e.
- $\to_e$ can be encoded as a term rewrite system (TRS):

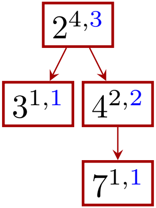
$$
\begin{aligned}
\mathrm{dep}(\mathrm{Tree}(\mathit{val}, \mathit{left}, \mathit{right})) &\to \mathrm{dep}(\mathit{left}) \\
\mathrm{dep}(\mathrm{Tree}(\mathit{val}, \mathit{left}, \mathit{right})) &\to \mathrm{dep}(\mathit{right})
\end{aligned}
$$

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

$\rightsquigarrow$ With monotone interpretations! [AProVE, 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));
}
```



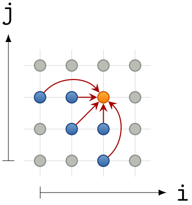
Monotone interpretation	Parallel complexity
$[\mathrm{dep}](x_1) = x_1$ $[\mathrm{Tree}](x_1, x_2, x_3) = x_2 + x_3 + 1$	$\lambda_t = \mathcal{O}(\mathrm{height}(t))$
$[\mathrm{dep}](x_1) = x_1$ $[\mathrm{Tree}](x_1, x_2, x_3) = \max(x_2, x_3) + 1$	$\lambda_t = \mathcal{O}(\mathrm{height}(t))$

What happens on regular 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: $[\mathrm{dep}](x_1, x_2) = x_1 + x_2$ $\quad \lambda_n \le 2n$

Same as in the polyhedral model!

Conclusion

Position:

- Automatic parallelization can take profit of monotonic interpretations.
- 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?