WCET Analysis of Code Parallelized with the Polytope Method

Björn Lisper
School of Innovation, Design, and Engineering
Mälardalen University

bjorn.lisper@mdh.se
http://www.idt.mdh.se/personal/blr/

2009-01-21

WCET Analysis of Parallel Programs

Much harder in general than for sequential programs (which is hard. . .)

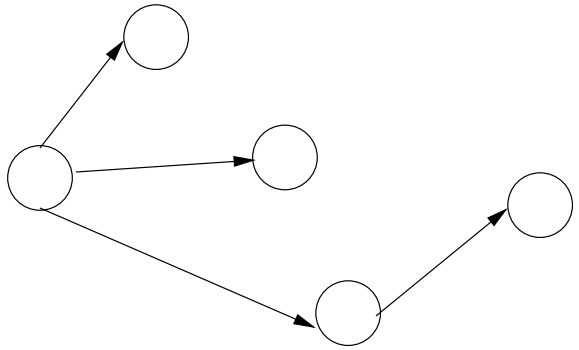
New factors that will affect the execution time, and make the analysis harder:

- communication
- synchronization
- Nondeterminism due to race conditions
- Possible deadlock (WCET $= \infty$)
- Unpredictable low-level timing due to sharing of HW resources

We'd better look for special niches, where one can do better

Static Parallelization

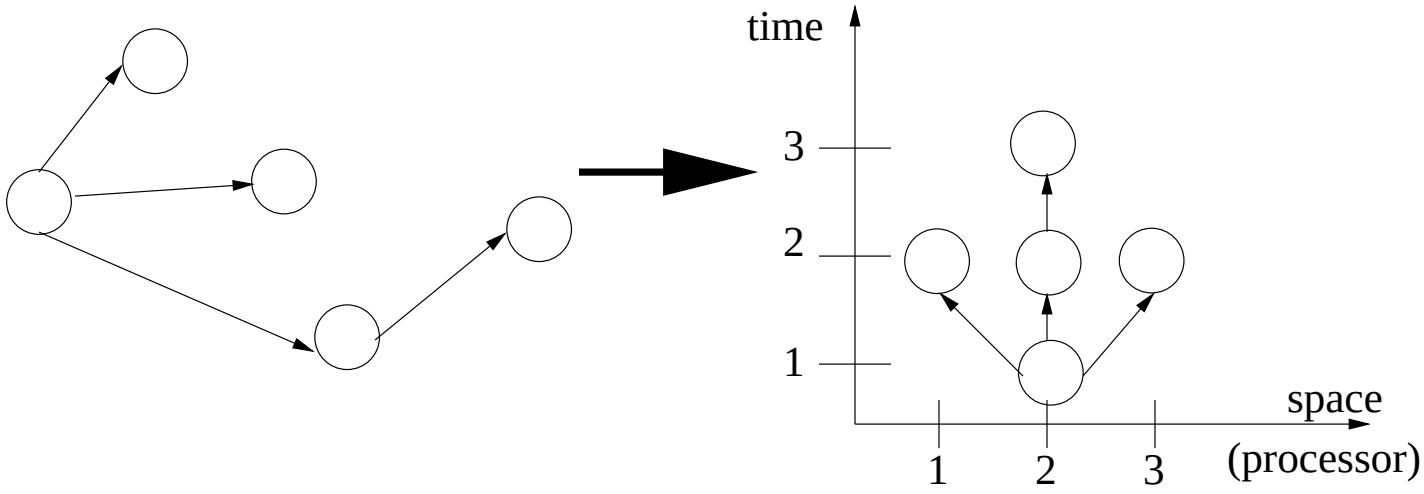
Computations that can be described by a fixed *data dependence graph*:



Nodes are tasks (in OR sense, not RT sense), dependences given by data transfers between tasks

Corresponds to straight-line code

If we *schedule* the graph in time, and *allocate* in space, then a parallel program can be synthesized:

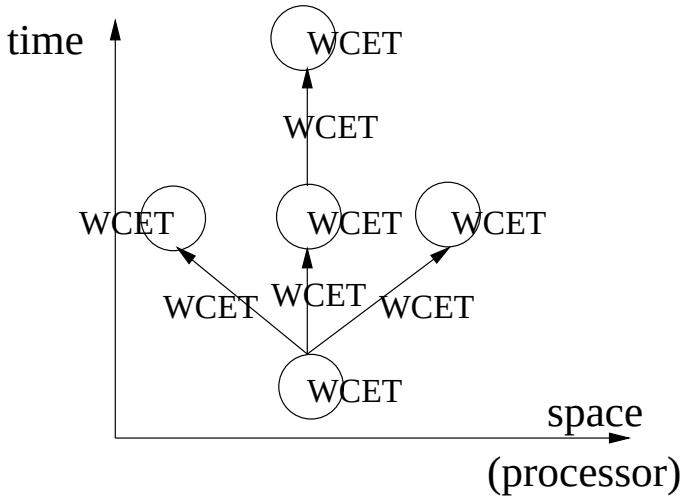


Data dependencies must be respected in time (causality)

Mapped computations may not overlap

Timing Analysis of Static Parallelization

If we can decide (bounds for) the execution times of the parts, then we can find bounds for the whole parallel computation



Similar to timing analysis of straight-line code, but communication time and possible interference through shared resources must be taken into account

The Polytope Method

Single-path code need not be straight-line code

Can also contain loops

The *Polytope Method* is a method to parallelize (possibly nested) loops

Works best for fairly regular loops: array computations, matrix codes etc.

Codes with largely data-independent program- & data flow, suitable for static scheduling (single-path code, or can be converted to such)

Possible “special niche”: WCET analysis of programs parallelized by the polytope method

The Polytope Method Step by Step

1. Determine the *iteration space* of the loop (one point per iteration)
2. Find the *dependencies* between different loop iterations
3. Schedule and allocate the loop iterations subject to the dependencies

Each point in the iteration space corresponds to an execution of the loop body

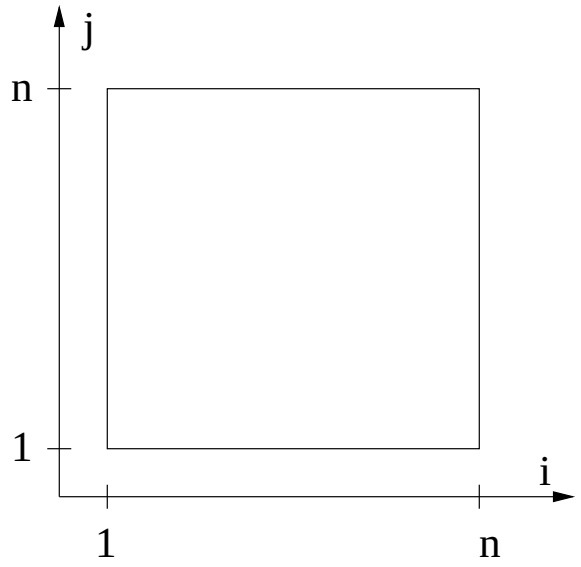
No dependencies between two executions $\Longrightarrow$ they can be executed in parallel

Example: a Loop and its Iteration Space

Matrix-vector multiply, $y = Ax$:

```
for i = 1 to n
  y[i] := 0
  for j = 1 to n
    y[i] := y[i] + a[i,j]*x[j]
```

Iteration space (inner loop):
$\{ (i, j) \mid 1 \leq i, j \leq n \}$



Each point corresponds to an execution of the loop body

Decide Data Dependencies

True dependencies uncovered by turning code into *single-assignment form*:

```
for i = 1 to n
  y[i,0] := 0
  for j = 1 to n
    y[i,j] := y[i,j-1] + a[i,j]*x[j]
```

`y[i,j]` produced at (i, j) in iteration space

`y[i,j]` assigned only once, then corresponds to a unique value (functional semantics)

Basically, the loop now specifies a mathematical *recursion equation*

Uniformization of the loop (also treat initializations as computations):

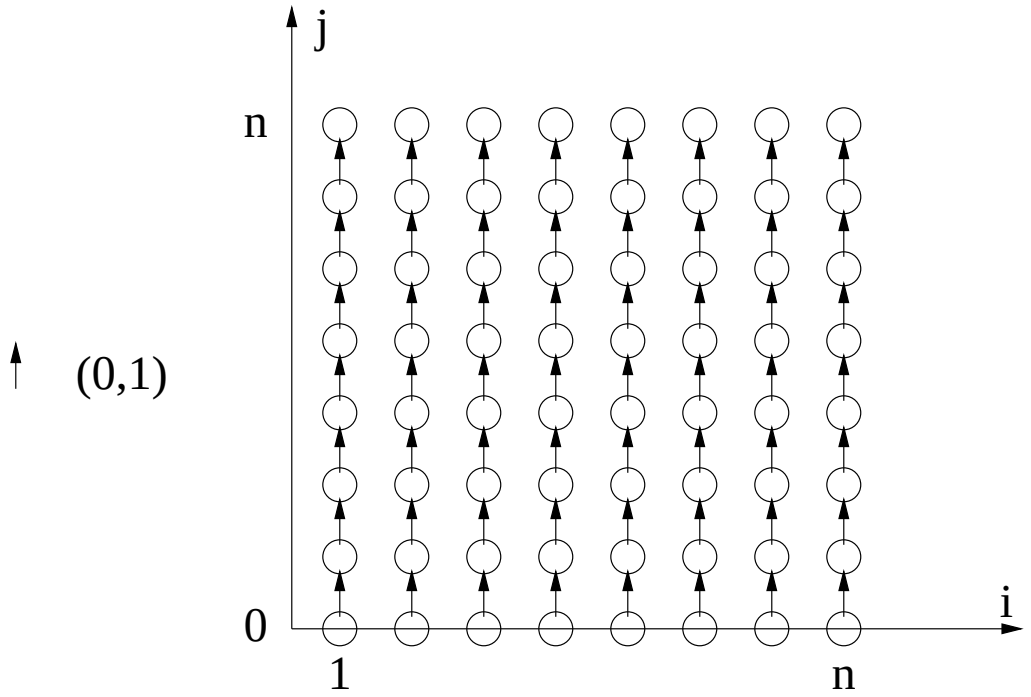
```
for i = 1 to n
  for j = 0 to n
    if j=0 then
      y[i,0] := 0
    else
      y[i,j] := y[i,j-1] + a[i,j]*x[j]
```

Computation of `y[i,j]` at (i, j) needs `y[i,j-1]` from $(i, j-1)$

Data dependence vector $(i, j) - (i, j-1) = (0, 1)$

This single vector characterizes *all* data dependencies between computations in the loop!

Data Dependence Vector



Scheduling and Allocation

Loop parallelization is now basically a task scheduling problem: each execution of the loop body is a task

Besides scheduling, we also must allocate tasks (loop body executions) to processors

Scheduling + allocation can be expressed as a *space-time mapping* from iteration space to a space-time with integral time t and a space of processor coordinates s:

$$\begin{pmatrix} s \\ t \end{pmatrix} = S(\vec{i}) = \begin{pmatrix} s(\vec{i}) \\ t(\vec{i}) \end{pmatrix}$$

Conditions on Space-Time Mappings

1. Data dependencies must be preserved in time, that is: $t(\vec{i}) > t(\vec{i'})$ if data dependence from $\vec{i'}$ to $\vec{i}$ (causality)
2. Space-time mapping is 1-1 (can be loosened to some extent)

Linear Space-Time Mappings

Linear mappings interesting due to simplicity, so let's use them to exemplify

Easy to check causality for them:

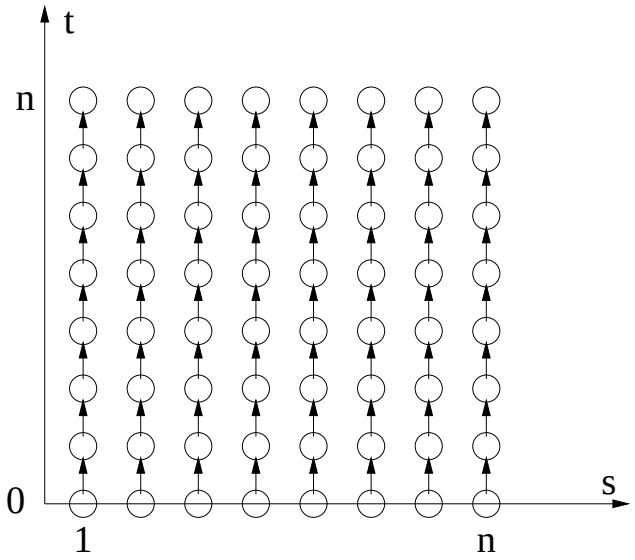
$$\begin{aligned} t(i,j) > t(i,j-1) \quad &\iff \quad t(i,j) - t(i,j-1) > 0 \\ &\iff \quad t((i,j) - (i,j-1)) > 0 \\ &\iff \quad t(0,1) > 0 \end{aligned}$$

Necessary and sufficient criterion: all data dependence vectors are mapped to positive time

A linear mapping S is 1-1 if invertible. Sufficient criterion: $\det(S) \neq 0$

A Possible Space-Time Mapping

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

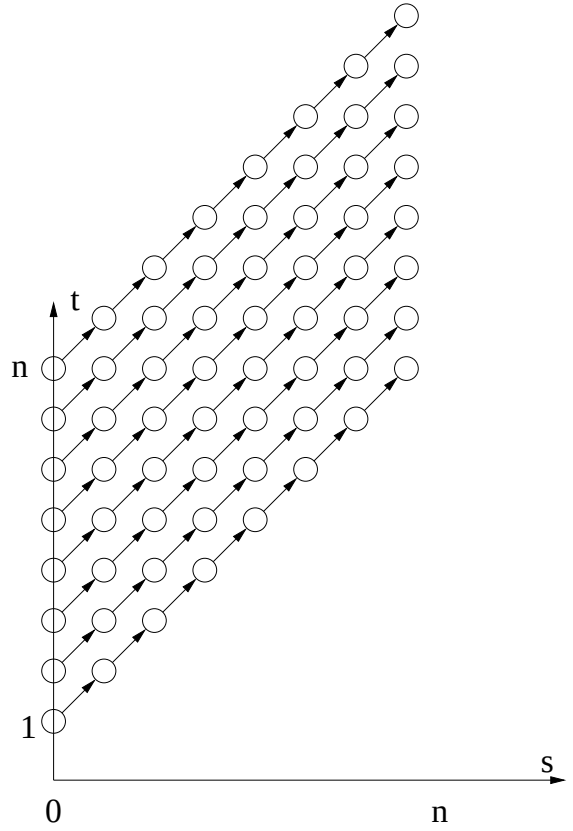


Yields in-place computation of each element of the resulting array, one element per processor (fine-grain parallelism)

Parallel code can be synthesized that implements the mapped computation

Another Space-Time Mapping

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$



Yields a pipelined computation, where computed elements move to the right

An Observation

This is a kind of static parallelization

Resulting parallel code should be as predictable as for the fixed data dependence graph

Each processor will execute some kind of single-path loop

Communication is also predictable

Should be possible to calculate WCET with high precision, provided the system is closed

Also by a measuring time for a single run, if execution time is independent of initial hardware state

Predicated Execution

Can do same “tricks” as for sequential programs to obtain parallel single-path code for loops with conditionally executed loop body parts:

```
for i = 1 to n
  for j = 1 to n
    if (dynamic condition) then ....
```

The index space will have “holes”. This transformation makes less regular loop programs parallelizable by the polytope method. Predicated parallel code can be generated according to the conditions defining the holes

Conclusions

High precision WCET estimation for some parallel programs seems possible

These are parallelized single-path loop programs

Must assume that the system is closed (no other activities), or sharing of resources may disturb the result

Can be applicable to control, media and signal processing, etc.

WCET by measuring should work just as for sequential single-path programs