Analysis and Optimization of Explicitly Parallel Programs using the Parallel Program Graph Representation

Vivek Sarkar

MIT Laboratory for Computer Science

(vivek@lcs.mit.edu)

Motivation

Current sequence of steps in compiling explicitly parallel/multithreaded programs:

1. Parallel program is mapped to low-level IL with library calls
2. Sequential compiler translates low-level IL to machine code
3. If machine code runs “correctly” then stop
4. Add volatile declarations to program and/or adjust compiler optimization options
5. Go back to step 1

Motivation (contd.)

Extension of sequential compiler analysis/optimization techniques to parallel programs is necessary for

- maintaining single-processor performance in parallel programs,
- adapting program parallelism to target parallel machine,
- and making compilation of parallel programs less tedious and less error-prone.

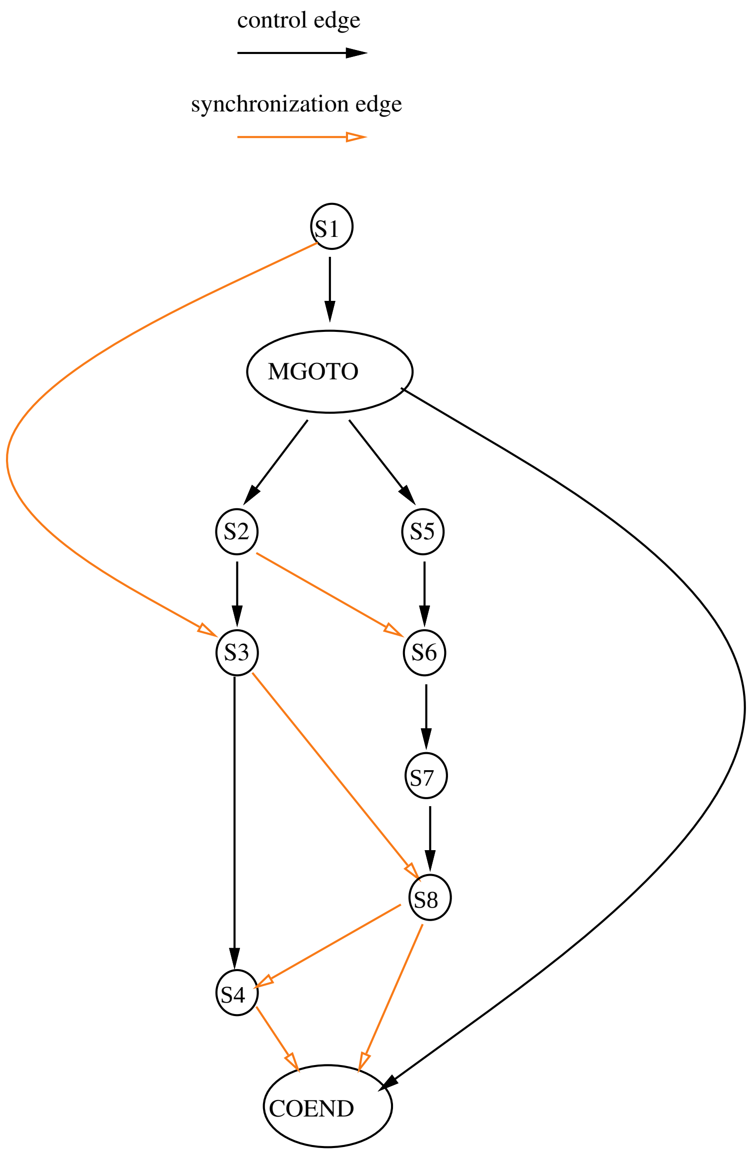
Parallel Program Graphs

A *Parallel Program Graph* $PPG = (N, E_{control}, E_{sync})$ is a directed multigraph consisting of:

- N, a set of nodes (includes CFG nodes and *mgoto* nodes). An *mgoto* node is used to create parallel threads of computation.
- $E_{control} \subseteq N \times N \times \{\text{TRUE}, \text{FALSE}, \text{UNCOND}\}$, a set of labeled *control* edges. Edge $(a, b, L) \in E_{control}$ identifies a control edge from node a to node b with label L.
- $E_{sync} \subseteq N \times N \times$ *SynchConds*, a set of *synchronization* edges. Edge $(a, b, f) \in E_{sync}$ defines a synchronization from node a to node b with synchronization condition f.

Example of a Parallel Program Graph

```
S1:  X₁ := ...
     post(ev1)
     cobegin
S2:    X₂ := ...
       post(ev2)
S3:    wait(ev1)
       post(ev3)
S4:    wait(ev8)
       X₄ := ...
    \\
S5:    ...
S6:    wait(ev2)
S7:    X₇ := ...
S8:    wait(ev3)
       post(ev8)
     coend
```



Relating CFGs to PPGs

Construction of PPG for a sequential program

- PPG nodes = CFG nodes
- PPG control edges = CFG edges
- PPG synchronization edges = empty set

Relating PDGs to PPGs

Construction of PPG from PDG:

- PPG nodes = PDG nodes

  (A region node in a PDG maps to an mgoto node in the PPG)

- PPG control edges = PDG control dependence edges

- PPG synchronization edges = PDG data dependence edges

  Synchronization condition f in PPG synchronization edge mirrors *context* of PDG data dependence edge

Reaching Definitions Analysis on CFGs

REACH$_{in}(n)$ = *set of definitions* d *s.t. there is a path from* d *to* n *and* d *is not killed along that path.*

$$\text{REACH}_{out}(n) \;=\; (\text{REACH}_{in}(n) - Kill(n)) \;\bigcup\; Gen(n)$$

$$\text{REACH}_{in}(n) \;=\; \bigcup_{p \,\in\, \text{pred}(n)} \text{REACH}_{out}(p)$$

Computing Redefinition (REDEF) sets for CFGs

REDEF$_{in}(n)$ = *set of definitions* d *s.t.* d *is redefined (killed) on ALL paths from* d *to* n *(and there is at least one path from* d *to* n*)*

$$\begin{aligned}\text{REDEF}_{out}(n) &= (\text{REDEF}_{in}(n) - Gen(n)) \bigcup \\ &\quad (Kill(n) \cap \text{REACH}_{in}(n)) \\ \text{REDEF}_{in}(n) &= \bigcap_{p \in \text{pred}(n)} \text{REDEF}_{out}(p)\end{aligned}$$

Three mutually exclusive cases for definition d and basic block n:

1. $d \in \text{REACH}_{in}(n)$
2. $d \in \text{REDEF}_{in}(n)$
3. there is no CFG path from d to n

Reaching Definitions Analysis on PPGs

$$\text{REACH}_{out}(n) = (\text{REACH}_{in}(n) - Kill(n)) \bigcup Gen(n)$$

$$\text{REACH}_{in}(n) = \left(\bigcup_{p \in \text{pred}(n)} \text{REACH}_{out}(p) \right) - \text{REDEF}_{in}(n)$$

$$\text{REDEF}_{out}(n) = (\text{REDEF}_{in}(n) - Gen(n)) \bigcup (Kill(n) \cap \text{REACH}_{in}(n))$$

$$\text{REDEF}_{in}(n) = \bigcup_{p \in \text{sync\_pred}(n)} \text{REDEF}_{out}(p) \bigcup \bigcap_{p \in \text{control\_pred}(n)} \text{REDEF}_{out}(p)$$

Reaching Definitions Analysis on Example PPG

Node n	REDEF$_{in}(n)$	REACH$_{in}(n)$	REDEF$_{out}(n)$	REACH$_{out}(n)$
S1	$\emptyset$	$\emptyset$	$\emptyset$	$\{X_1\}$
cobegin	$\emptyset$	$\{X_1\}$	$\emptyset$	$\{X_1\}$
S2	$\emptyset$	$\{X_1\}$	$\{X_1\}$	$\{X_2\}$
S3	$\{X_1\}$	$\{X_2\}$	$\{X_1\}$	$\{X_2\}$
S5	$\emptyset$	$\{X_1\}$	$\emptyset$	$\{X_1\}$
S6	$\{X_1\}$	$\{X_2\}$	$\{X_1\}$	$\{X_2\}$
S7	$\{X_1\}$	$\{X_2\}$	$\{X_1, X_2\}$	$\{X_7\}$
S8	$\{X_1\}$	$\{X_2\}$	$\{X_1, X_2\}$	$\{X_7\}$
S4	$\{X_1, X_2\}$	$\{X_7\}$	$\{X_1, X_2, X_7\}$	$\{X_4\}$
coend	$\{X_1, X_2, X_7\}$	$\{X_4\}$	$\{X_1, X_2, X_7\}$	$\{X_4\}$

Related Work

- [Midkiff et al 89], [Chow, Harrison 92], ...

  Analysis of explicit (nondeterministic) parallel programs with scalar and array variables, a sequentially consistent memory model, and structured parallelism (no explicit synchronization)

- [Pingali et al 90]

  Presented a constant propagation algorithm for Dependence Flow Graphs (dataflow graphs with an imperative store)

- [Srinivasan 94], [Ferrante et al 96]

  Analysis of explicit deterministic parallel programs with scalar and array variables and copy-in/copy-out semantics

Conclusions

In this talk, we

- motivated using the Parallel Program Graph (PPG) representation in analysis and optimization of parallel programs, and

- presented a solution for reaching definitions analysis on PPGs that is more precise than in past work.

Future Work

- Extend other traditional analysis and optimization algorithms for use on PPGs

- Use PPGs as intermediate representation in common compilation and execution environment for different parallel programming languages

- Extend PPG execution model to support mutual exclusion and nondeterminism

Parallel Control Flow in a PPG

Three cases:

1. **An mgoto node** a **with** $k \geq 0$ **outgoing control edges,** (a, b_1, UNCOND)**, ...,** (a, b_k, UNCOND)**, all with label** UNCOND**:**

   An execution instance I_a of node a creates new execution instances $I_{b_1}, \ldots, I_{b_k}$ of nodes $b_1, \ldots, b_k$ and then terminates itself.

2. **A non-mgoto node** a **with one outgoing control edge** (a, b, L) **for branch label** L**:**

   When an execution instance I_a of node a evaluates node a's branch label as L, it creates a new execution instances I_b of node b and then terminates itself.

3. **A non-mgoto node a with no outgoing control edges for branch label L:**

   When an execution instance I_a of node a evaluates node a's branch label as L, it terminates itself.

Execution Histories

$$H(I_a) = \text{execution history of instance } I_a \text{ of PPG node } a$$
$$= \text{sequence of (node,label) branch conditions that caused execution instance } I_a \text{ to be created}$$

Execution histories are defined recursively:

1. $H(I_{start}) = <>$ (empty sequence)

2. If execution instance I_a creates execution instance I_b due to control edge (a, b, L), then $H(I_b) = \text{concat}(H(I_a), < a, L >)$

Synchronization Conditions

Consider synchronization edge $(a, b, f) \in E_{sync}$

Synchronization condition f is a boolean function on execution histories

Given execution instances I_a and I_b of nodes a and b, $f(H(I_a), H(I_b)) = true$ means that execution instance I_a must complete execution before execution instance I_b can be started

Weak (Deterministic) Memory Consistency Model

- All memory accesses are assumed to be non-atomic
- *Read-write hazard* — if I_a reads location l in σ_i and there is a parallel write of a different value, then the result is an error
- *Write-write hazard* — if I_a writes value v into location l and there is a parallel write of a different value, then the resulting value in location l is undefined
- Separation of data communication and synchronization:
  - Data communication specified by read/write operations
  - Sequencing specified by synchronization and control edges

Control-Independent Synchronizations in a PPG

A synchronization edge (x, y, f) is *control-independent* if a necessary condition for $f(H(I_a), H(I_b))$ =*true* is that *nodeprefix*$(H(I_x), a)$ = *nodeprefix*$(H(I_y), a)$ for all nodes a that are control ancestors of both x and y.

Control-Independent Synchronizations in a PPG (contd.)

Consider an execution instance I_x of PPG node x with execution history $H(I_x) =< u_1, L_1, \ldots, u_i, \ldots, u_j, \ldots, u_k, L_k >$, $u_k = x$. We define *nodeprefix*$(H(I_x), a)$ as follows:

If $a \neq x$, *nodeprefix*$(H(I_x), a) =< u_1, L_1, \ldots, u_i, L_i >, u_i = a$, $u_j \neq a$, $i < j \leq k$.

If $a = x$, then *nodeprefix*$(H(I_x), a) =$ $H(I_x) =< u_1, L_1, \ldots, u_i, \ldots, u_j, \ldots, u_k, L_k >$.

A node a is a *control ancestor* of node x if there exists an acyclic path of control edges from *START* to x such that a is on that path.