Overview

MRIS Problem

MRIS – minimum instruction scheduling

MTIS – minimum time instruction scheduling

RCMTIS – register-constrained minimum time instruction scheduling

SMRTIS – simultaneous minimization of space and time

Code Generation — Phases

Phase ordering problems

MRIS problem

FROM THE MRIS PROBLEM TO INTEGRATED CODE GENERATION
Contiguous schedules

A contiguousschedule of a DAG $S = (V, E)$ is a schedule where all descendants of one child are listed before any of the remaining descendants of the other one.

A contiguous schedule $S$ of a DAG $G = (V, E)$ is a schedule such that all children of a node $v$ are listed before $v$.

\[ V = V_1 \cup V_2 \]

wheredefinition: label

Example:

For a binary node $v$, label $v = \max \{ (\alpha(x)) \mid x \in \text{L}(v) \}$

For a unary node $v$, label $v = \max \{ (\alpha(x)) \mid x \in \text{L}(v) \}$

For any leaf $v$, label $v = 1$.

\[ V = V_1 \cup V_2 \]

Theorem:

The minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$ equals the minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$.

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Proof by induction:

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

What about DAGs?

Computing a space-optimal schedule for a DAG is NP-complete.

Extending the labeling algorithm to DAGs (computing common subexpressions only once)

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Sethi-Ullman labeling algorithm for trees

The Sethi-Ullman labeling algorithm for trees [Sethi, Ullman J.ACM 1970] visits at each binary node the subtree with larger label value first.

Theorem:

The minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$ equals the minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$.

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Proof by induction:

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

What about DAGs?

Computing a space-optimal schedule for a DAG is NP-complete.

Extending the labeling algorithm to DAGs (computing common subexpressions only once)

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Sethi-Ullman labeling algorithm for trees

The Sethi-Ullman labeling algorithm for trees [Sethi, Ullman J.ACM 1970] visits at each binary node the subtree with larger label value first.

Theorem:

The minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$ equals the minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$.

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Proof by induction:

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

What about DAGs?

Computing a space-optimal schedule for a DAG is NP-complete.

Extending the labeling algorithm to DAGs (computing common subexpressions only once)

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Sethi-Ullman labeling algorithm for trees

The Sethi-Ullman labeling algorithm for trees [Sethi, Ullman J.ACM 1970] visits at each binary node the subtree with larger label value first.

Theorem:

The minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$ equals the minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$.

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Proof by induction:

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

What about DAGs?

Computing a space-optimal schedule for a DAG is NP-complete.

Extending the labeling algorithm to DAGs (computing common subexpressions only once)

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Sethi-Ullman labeling algorithm for trees

The Sethi-Ullman labeling algorithm for trees [Sethi, Ullman J.ACM 1970] visits at each binary node the subtree with larger label value first.

Theorem:

The minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$ equals the minimum number of registers needed by any spill-free schedule of the sub-tree rooted at $v$.

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

Proof by induction:

\[ \text{Lab}(v) = \text{Lab}(v_1) \]

\[ \text{Lab}(v) = \text{Lab}(v_1) \]
There are three possibilities to split the DAG.

Example: $\beta$, $0$, $1$.

Observation: Different bifectors may yield the same schedule...

We start a postorder traversal of $G$. For each of the two different bifectors $p$, $q$...

A decision node is a binary node which is not a free node.

A free node is either a leaf or an inner node.

A decision node with two children is an inner node.

A leaf node is a terminal node.

We schedule all the free nodes in the DAG with decision nodes $1$, $2$, $3$, $4$, $5$.

First improvement: Traverse only bifectors that yield different schedules:

Find an optimal contiguous schedule for DAGs [K., Rauber 99/95] (2)
Observations:

- Correspond to a schedule of scheduled nodes and vice versa.
- Labels on a path from top to node z are the same set scheduled(c) or DAG nodes has been scheduled.
- All paths to instances of the same zero-indeg set have equal length.
- In general, multiple instances of same zero-indeg set.

Recall: List scheduling = Local scheduling by topological sorting.

Graph-based method: Naive enumeration, Selection Tree (1)

From Selection Tree to Selection DAG (1)

ADVANCED COMPILER CONSTRUCTION—MRIS problem.


Page 15

Page 16
First nonempty \( \frac{1}{n} \) has been reached

+ Stop construction as soon as
+ need not search entire \( \frac{1}{n} \).

level-wise = cumulative

level-wise: The predecessors \( z \) of a selection node \( z \in \frac{1}{n} \) are located in \( \frac{i}{n} \).

Lemma: If \( z \) is optimal for \( \frac{i}{n} \) with \( S \) and \( S \), registers \( \pi \) that correspond to

staged: For each selection edge, \( \pi \) encoding in same zero-indeg set

Level-wise construction of the selection DAG

The schedule \( S \) is optimal if \( \forall z \in \frac{i}{n} \).

Hence we may choose any path \( \pi \).

\{ (z) \} = \{ (z) \} \not= \{ (z) \} \not= \{ (z) \}.

The same values reside in registers, namely:

The corresponding paths \( \pi \) and \( \pi \) in \( \frac{i}{n} + 1 \) levels corresponding to \( z \).

Proof (check): For each selection DAG path \( \pi \) ending in the same zero-indeg set

From Selection Tree to Selection DAG (3)

For all schedules \( S \) for the same zero-indeg set \( z \), the corresponding paths \( \pi \) and \( \pi \) in the same node.

For all schedules \( S \) for the same zero-indeg set \( z \), the same values reside in registers, namely:

The same values reside in registers, namely:

The same values reside in registers, namely:

The corresponding paths \( \pi \) and \( \pi \) in the same node.

From Selection Tree to Selection DAG (2)

For all schedules \( S \) for the same zero-indeg set \( z \), the corresponding paths \( \pi \) and \( \pi \) in the same node.

For all schedules \( S \) for the same zero-indeg set \( z \), the same values reside in registers, namely:

The same values reside in registers, namely:

The corresponding paths \( \pi \) and \( \pi \) in the same node.

From Selection Tree to Selection DAG (1)
Modified algorithm

\[ L_{ik} \rightarrow \text{newList} \]

\[ z_0 \]

\[ \text{for } k = 1 \text{ to } \text{MAXREG} \text{ do} \]

\[ \text{for } L_{ik} \text{ from } 0 \text{ to } n \text{ do} \]

\[ \forall z \in L_{ik} \text{ do} \]

\[ \forall v \in z \text{ do} \]

\[ z = z \setminus v; \]

\[ \text{mreg.needof} \quad S(z); \]

\[ \text{if } L_{ik} < 1 \text{ then } \text{lookup} z \text{ endif} \]

\[ \text{if } m_k / \max \text{ then } \text{remove} z \text{ endif} \]

\[ \text{if } L_{ik} < 1 \text{ then } \text{insert} z \text{ endif} \]

\[ \end{verbatim} \]

---

**Summary MRIS (2)**

- **MRIS** is NP-complete for DAGs
- **4 algorithms:**
  - Sethi-Ullman labeling
  - K_dynamic programming method
  - KHeuristic register bound
  - Lineage fusion
  - Sequencing by listscheduling
  - No backtracking
  - Goodheuristic
- **Why searching for an optimal solution?**
  - Scientific interest
  - Feedback to processor design
  - Aggressive space optimization may pay off for embedded systems
  - Quantitative assessment of the quality of fast heuristics
- **Experimental results:**
  - Parallelizable up to 40-50 nodes
  - Practical up to 40 nodes
  - Good heuristics
  - Even slightly better than MIPS R10K native compiler on SPEC95

---

**Model algorithm**

```plaintext
; parallelizable + practical up to 40-50 nodes + good heuristics + early as possible
```

---

**MRIS – heuristic solution**

- Groundtruth in all lineage method
- Optimal schedule for DAGS
- K_dynamic programming method
- Optimal continuous schedule for DAGS
- KHeuristic register bound
- Seth-Ullman labeling

---

**MRIS – space-optimal scheduling**

Even slightly better than MIPS R10K native compiler on SPEC95

---

**MRIS (2) – searching for an optimal solution?**

- Good heuristic
- No backtracking
- Sequencing by listscheduling
- Finding lineage
  - Is passed from Instruction to Instruction
  - Instruction lineage = sequence of instructions in which a single register
  - See Amaral's nice presentation material, link on course page
Integrated code generation

Phase ordering problem: Register allocation ⇔ Instruction scheduling

- May destroy quality of a previously good schedule
- Schedule must be scheduled as well

The-scheduled MIR/LIR or target code with symbolic registers

Determining the ranges requires a linear sequence of instructions

Integrating instruction selection

Kästner’s ILP formulation

Our dynamic programming method

Goodman/Hsu’88

Heuristics

Alternating iterative methods

Phase ordering problem: Instruction allocation ⇔ Initial scheduling

Spill code must be scheduled

Scheduling determines the ranges ⇔ Interferences

Example:

(a) Register allocation after scheduling

(b) Register allocation before scheduling

Determining live ranges requires a linear sequence of instructions

Spill code must be scheduled as well

Integrating instruction selection

Kästner’s ILP formulation

Our dynamic programming method

Goodman/Hsu’88

Heuristics

Alternating iterative methods

Phase ordering problem: Instruction allocation ⇔ Initial scheduling

Spill code must be scheduled

Scheduling determines the ranges ⇔ Interferences

Example:
Recall: Greedy list scheduling for VLIW architectures

Extended selection node \( z \) summarizes all schedules of \( \text{scheduled}(z) \) that end with time profile \( (t, P) \).

Problems with simultaneous space and time minimization

Recall: Greedy list scheduling for VLIW architectures

Extended selection node \( z \) summarizes all schedules of \( \text{scheduled}(z) \) that end with time profile \( (t, P) \).

Problems with simultaneous space and time minimization

Recall: Greedy list scheduling for VLIW architectures
Algorithm is practical.

Time-space profiles of order of construction:

- Structuring of search space and dependence analysis.
- Exploring domain-specific properties.
- Dynamic programming.
- Combination with decision tree.
- Topological sorting (list scheduling).
- Algorithmic techniques used.

Optimal solution of MRS includes WS/MPS/ROGTMS.

---

Time-space profiles for non-homogeneous register sets

Extractions (1)

Order of construction of the \( L' \) according to desired optimization goal.

Partition lists \( L' \) in sublists \( L'_{\text{time}} \) of \( \text{MAXDELAY} \).

We observe: By appending a DAG node to a schedule, we have

\[ \text{time} + 1 \geq \text{time}(s) \geq \text{time}(\varepsilon) \geq (s) \geq \{a\} \geq \{b\} \]

---

Simultaneous space and time optimization with time profiles

---

Theorem: After the optimization based on integer linear programming, all time profiles with same zero in-set are comparable.

---

Optimal solution of MRS

---

Example

---

[\( \text{BEDNARSKI'01} \)]
Optimal scheduling of spill code [K.00]

Heuristics for pruning of large selection DAGs

[Bisectortechnique: usually worse results than random scheduling [K.00]

(Limited) admission of recomputations

Ex.class of DAGs: linear decrease of \( m \rightarrow \exp \) time \((S)\)