SOFTWARE PIPELINING

[Charlesworth'81] [Rau/Glaeser'81] [Aiken/Nicolau'88] [Lam'88]...

as also called "moduloscheduling"

+ for loops, also with loop-carried dependences
+ profitable especially for long pipeline delays, multiple units
+ to increase instruction-level parallelism
+ for loops, also with loop-carried dependences
+ basic ideas: overlap instructions from different iterations

Goal:
Combine operations from different subsequent iterations
such that no data dependences are violated
and no resource conflicts occur.

Total execution time:

\[ \text{length(prologue)} + \text{length(epilogue)} + \text{length(kernel)} \]

The kernel is called the "kernel" (also called "stable state")
and forms a new "kernel" (also called "stable state")

Initiating unrolling isn't realistic...

Optimal solution: NP-complete (as usual)...

1. Prologue
2. Pattern
3. Kernel
4. Epilogue

Software pipelining

Minimize: \( \text{III} \)

Minimize \( \text{III} \) = number of instructions in the original loop body

\[ \text{III} = \text{length(prologue)} + \text{length(epilogue)} + \text{length(kernel)} \]

\[ \text{Total execution time} = \text{length(prologue)} + \text{length(epilogue)} + \text{length(kernel)} = \text{cycles between issuing the first instructions of two subsequent kernel iterations} \]

Optimize: \( \text{III} \) = # iterations (kernel)...

Optimum: \( \text{III} \), \( \text{II} \)
Lower bounds for the minimum initiation interval \( MII \):

\[
MII = \text{number of instructions in the original loop body} / \text{number of functional units/processors}
\]

\[MII = \max \left( \text{Res}MII, \text{Rec}MII \right)\]

Resource constraints:

\[
\text{Res}MII = \frac{\text{cell} \times \text{cycle}}{\text{distancesum}}
\]

Scheduling

Consequences for the register need

Liveranges may span over multiple iterations → high register need

If not possible, increase \( MII \) and try again.

Mark occupied slots in all iterations.

Apply some placement heuristic, e.g.: as early as possible.

If there are recurrences in the dependence graph:

Apply some local scheduling heuristic, e.g.: list scheduling.

Mark occupied slots in all iterations...

If not possible, increase \( MII \) and try again...

Good heuristic: e.g. "Hypernode Reduction Module Scheduling" + disadvantage: separate schedules of SCCs may not fit well together

Similarly: collapse "then-do" statements

(d) apply list scheduling to resulting acyclic graph

(c) schedule each SCC individually

(b) find cycle in SCC with longest accumulated distance

(a) find cycle in program data dependence graph

Concerning data dependencies:

Due to (cycle chains of data dependence (DCC))

\[\{ \text{begin with Start} \} \text{ delay sum over } C \text{ distance sum over } C \]

Scheduling the kernel (2)

Module resource reservation table

+ Modularesourcereservationtable

+ If there are recurrences in the dependence graph:

+ Moduloresource reservation table

Similar: collapse

if..then..else

statements

+ disadvantage: separate schedules of SCCs may not fit well together

+ disadvantage: separate schedules of SCCs may not fit well together

Scheduling the kernel

Consequences for the register need

Liveranges may span over multiple iterations → high register need

If not possible, increase \( MII \) and try again.

Mark occupied slots in all iterations.

Apply some placement heuristic, e.g.: as early as possible.

If there are recurrences in the dependence graph:

Apply some local scheduling heuristic, e.g.: list scheduling.

Mark occupied slots in all iterations...

If not possible, increase \( MII \) and try again...

Good heuristic: e.g. "Hypernode Reduction Module Scheduling"

\[\{ \text{begin with Start} \} \text{ delay sum over } C \text{ distance sum over } C \]

Scheduling the kernel (2)

Module resource reservation table

+ Modularesourcereservationtable

+ If there are recurrences in the dependence graph:

+ Moduloresource reservation table

Similar: collapse

if..then..else

statements

+ disadvantage: separate schedules of SCCs may not fit well together

+ disadvantage: separate schedules of SCCs may not fit well together

Scheduling the kernel

Consequences for the register need

Liveranges may span over multiple iterations → high register need

If not possible, increase \( MII \) and try again.

Mark occupied slots in all iterations.

Apply some placement heuristic, e.g.: as early as possible.

If there are recurrences in the dependence graph:

Apply some local scheduling heuristic, e.g.: list scheduling.

Mark occupied slots in all iterations...

If not possible, increase \( MII \) and try again...

Good heuristic: e.g. "Hypernode Reduction Module Scheduling"

\[\{ \text{begin with Start} \} \text{ delay sum over } C \text{ distance sum over } C \]
Extensions

Hardware support: Rotation register files (cyclic register remapping)

Software pipelining for loops with conditional branches

Optimal unrolling factor [Badia/Sanchez/Cortadella'96]

Formulation as ILP possible (feasible for $N \approx 25$) [Gao'96]

Optimal software pipelining is NP-complete

Further reading on resource-constrained software pipelining:


[Sanchez'96] PhD thesis, UPc Barcelona

Loop unrolling may provide additional efficiency

[Stoodelf/Lee'96]