

Java Byte Code Scheduling Based on the Most-Often-Used-Paths in Programs with Branches

Eryk Laskowski¹, Marek Tudruj¹, Richard Olejnik², Bernard Tournel²

¹ *Institute of Computer Science
PAS, Warsaw, Poland
{tudruj, laskowsk}@ipipan.waw.pl*

² *Laboratoire d'Informatique
Fondamentale de Lille (LIFL)
{olejnik, tournel}@lifl.fr*

Contents

1. Introduction, motivation of our work
2. Program representation
3. Optimization algorithms
4. Example
5. Summary

Introduction

- The paper presents Java program optimization methods, which transform the byte-code of multithreaded sequential Java programs into parallel versions distributed in a set of JVMs, which reduce program execution time.
- Motivation: no sufficient care has been paid so far to pre-optimisation support of placement of distributed Java programs on JVMs before execution – it can result in an unbalanced execution of programs at run time.

Overview of proposed solution

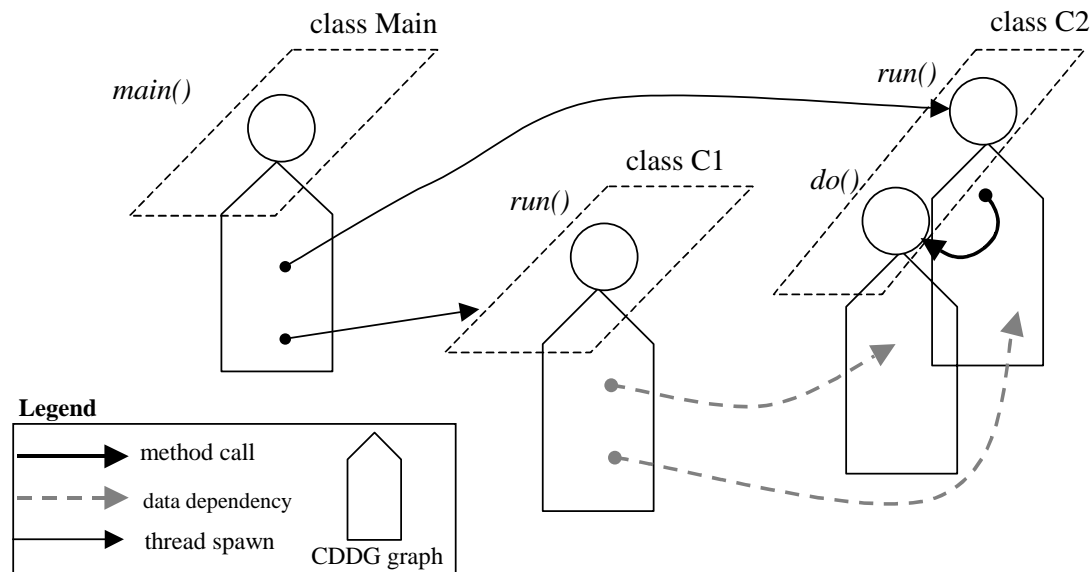
- Static pre-optimization algorithm applied to the byte-code generated by the Java compiler.
 - construction of a data flow graph of the Java program based on data and control dependencies in its byte-code,
 - a fine grain analysis of the data flow graph, the outcome is a macro data flow graph (MDFG).
 - transformation into a parallel version distributed on a set of processors (JVMs) (clustering and scheduling).
- The result –reduced inter-processor data communication and balanced processor computation load.

Program representation

- Optimization algorithms use program dependence graphs generated by:
 - fine grain analysis of a byte-code and execution traces for representative data sets,
 - analysis of execution traces, collected by profiling program runtime behavior for sets of representative data.
- Program representations:
 - Method/Thread Call Graph (MTCG),
 - Control/Data Dependence Graph (CDDG),
 - Macro Data Flow Graph (MDFG).

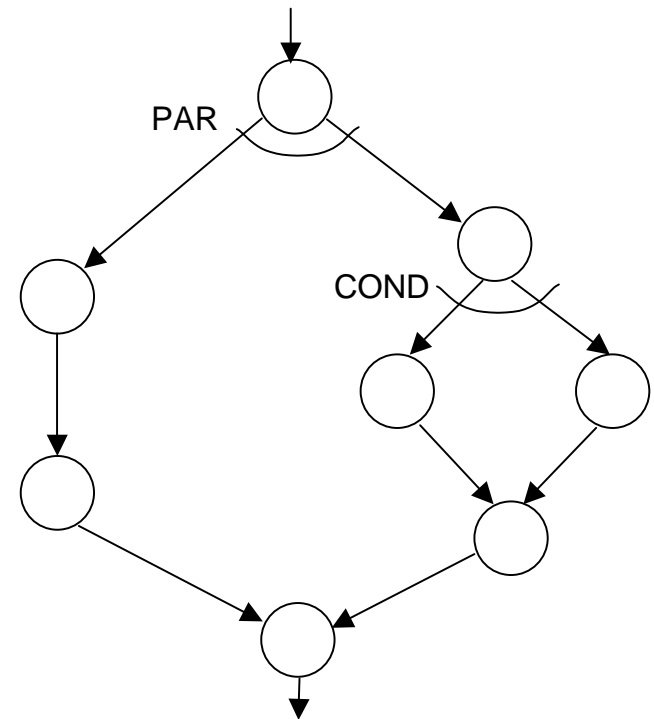
MTCG and CDDG graphs

- MTCG graph:
 - methods are shown as nodes, their mutual calls are shown as edges,
 - each called method has its own control/data dependence graph (CDDG).
- CDDG graph:
 - nodes correspond to sequences of byte code instructions inside a method,
 - edges correspond to control/data flow between byte code instructions of a method.



Macro Data Flow Graph (MDFG)

- MDFG graphs are obtained by the transformation of the MTCG (agglomeration of sequential instructions that appear between instructions that can potentially produce parallelism, i.e. inter-method calls, thread spawning and inter-object data references).



Program optimization algorithms

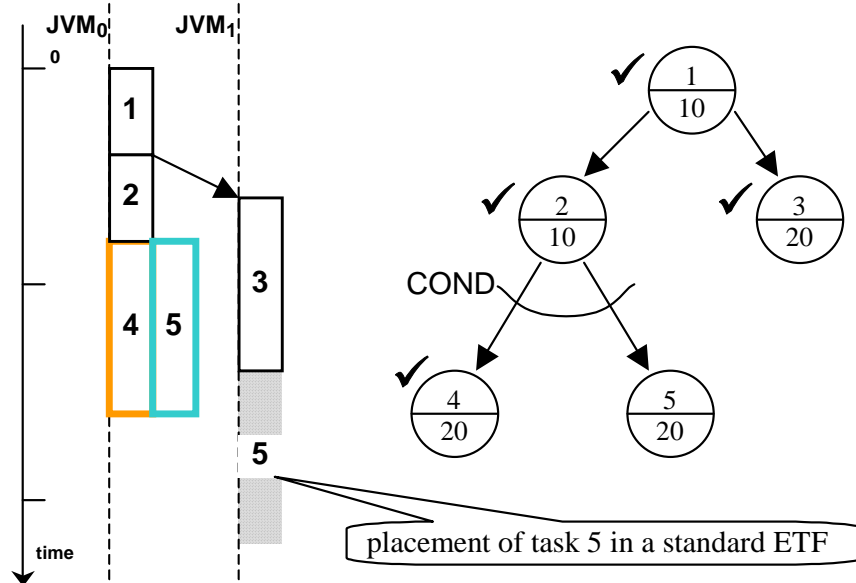
- Two step approach:
 - 1) apply a clustering algorithm based on the dominant sequence approach (DSC) applied to unlimited number of processors
 - 2) perform a mapping algorithm which assigns clusters to the real number of physical JVMs with load balancing.
- JavaParty is used for implementation of the schedule on JVMs.

The clustering phase

- Clustering to an unlimited number of processors.
- Modified clustering algorithms based on a DSC heuristics extended by *trace scheduling* approach and *most-often-used-path* optimization method:
 - at each clustering step, the algorithm selects a node with the highest priority from the set of *ready nodes*,
 - the algorithm selects outgoing paths of a recently visited branch macro node in the descending order of their probabilities.

The mapping phase

- The mapping algorithm is based on list scheduling with modified earliest task first (ETF) heuristics.
- The algorithm uses *detection of mutually-exclusive paths* optimization method:



Dynamic optimization of program execution

- MDFG and MTCG graphs allow to perform static optimizations, which may not be sufficient for a efficient program execution in the case of
 - irregular applications,
 - dynamic modifications of resource availability.
- A heuristic is proposed (ADAJ) that aims to detect unbalanced load situations and to correct them.

Example - pseudocode of Java program

- Two parallel tasks (TASK1 and TASK2), which are spawned from the main method of the program (MAIN),
- each task performs matrix addition and then spawns two threads for parallel execution of matrix multiplication,
- depending of the value of the conditional variable (**flag**), matrix addition and multiplication operate on different input matrices.

MAIN

do in parallel

TASK 1

if flag == true **then**

A = A + B

do in parallel

Res1 = A x C

Res2 = B x D

else

A = A + C

do in parallel

Res1 = A x B

Res2 = C x D

endif

TASK 2

if flag == true **then**

E = E + F

do in parallel

Res3 = E x G

Res4 = F x H

else

E = E + G

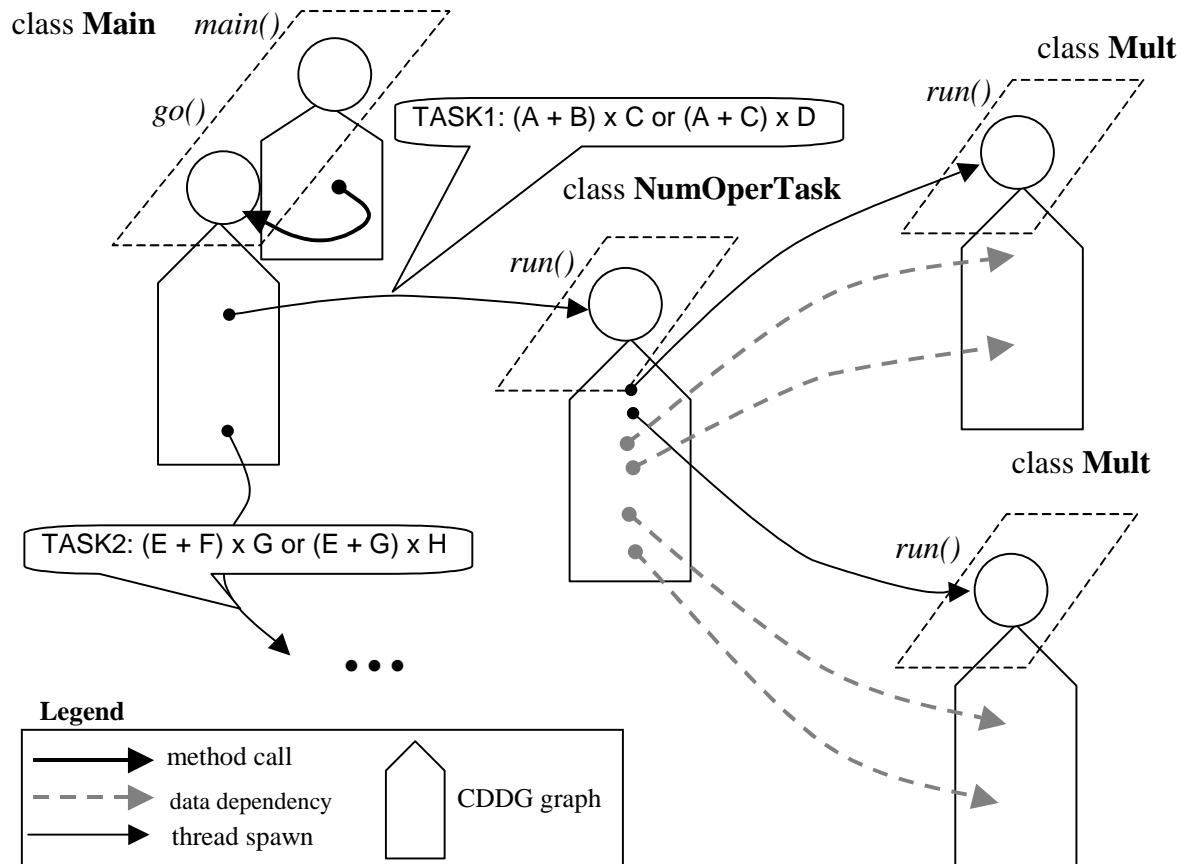
do in parallel

Res3 = E x F

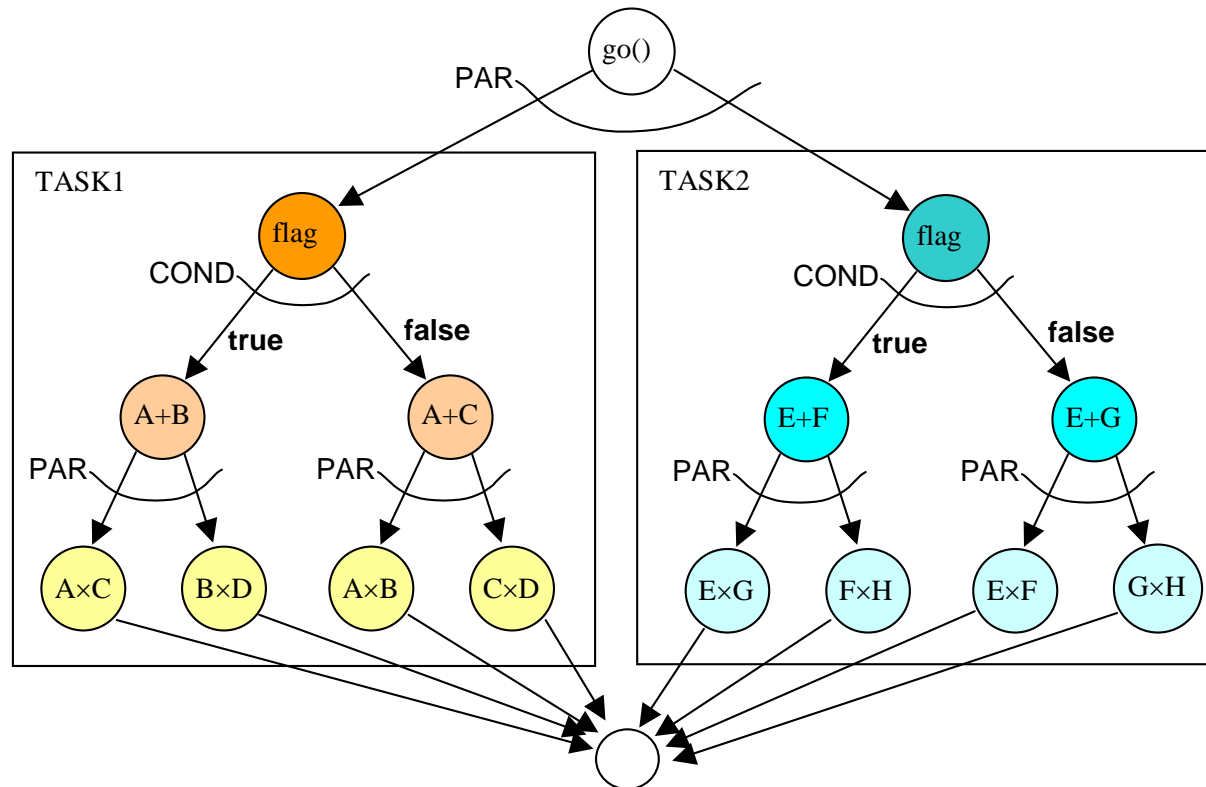
Res4 = G x H

endif

Example - the MTCG graph

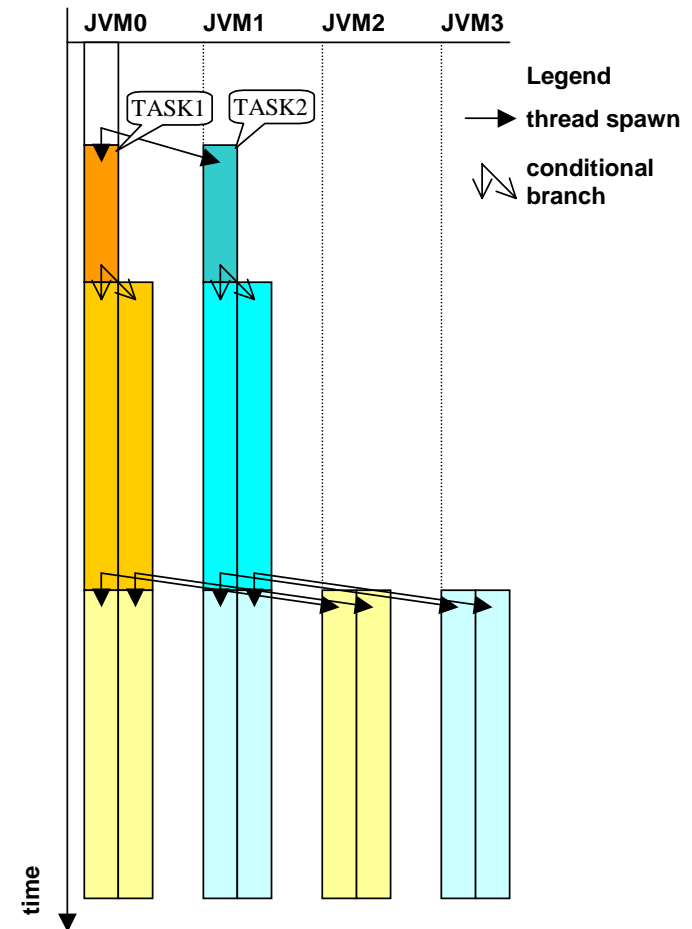


Example - the result of the clustering phase



Example - the result of the mapping phase

- Macro nodes of the MDFG have been scheduled onto a system of JVMs.
- Macro nodes from mutually exclusive conditional paths are mapped in parallel onto the same JVM (since it is impossible to execute them in the same time).



Summary

- Scheduling algorithms for macro data flow graphs generated from the byte code of Java programs have been proposed.
- The algorithms use trace-based approach to scheduling of conditional nodes and loops in the graphs. The obtained pre-scheduled Java programs, distributed over a set of JVMs, aim in reduction of program execution time.
- The implementation of the optimization algorithms is under development currently.