Improved Execution Migration

IIT Bombay

February 19, 2018

1. Execution Migration Introduction

2. Suggested Improvements
   - Simultaneous transformation
   - Migration at loop branches
   - Single frame transformation
   - Live variables

Introduction

- Venkat et al[1] show energy efficiency and performance gain by Heterogenous ISA CMPs.
- DeVuyst et al[2] explore cost-effective migration strategies.
  - Global variables kept common during compile-time.
  - Heap memory only touched by *malloc* function, same implementation used.
  - Keep functions at same virtual address for same pointers.
  - Stack frame size and local variable location maintained same at compile time by using padding.
  - **Dynamic modification:** Hard-coded stack offset in instructions, function arguments, stack object pointers.
- DeVuyst et al look at only function calls as potential migration points, and suggest using binary translation for immediate shift until a migration point is reached.

[1] A.Venkat, D.Tullsen - Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor

[2] M.DeVuyst et al - Execution Migration in a Heterogeneous-ISA Chip Multiprocessor

Simultaneous transformation

- Stack transformer modifies entire stack at migration time.
- Only last frame on the stack (current function) is currently in use.
- Start transformation of other frames simultaneously on other processor before migration point is reached.
- Cost of migration reduces to transformation cost of one frame

Migration at loop branches

- As seen in graphic, some programs spend a lot of time inside loops.
- Loop branches (jmp instruction to loop start label) will be present in both ISA.
- Migration at this point will be able to harness efficiency for further loop iterations.
- Binary translation not necessary because migration points will be much more frequent.

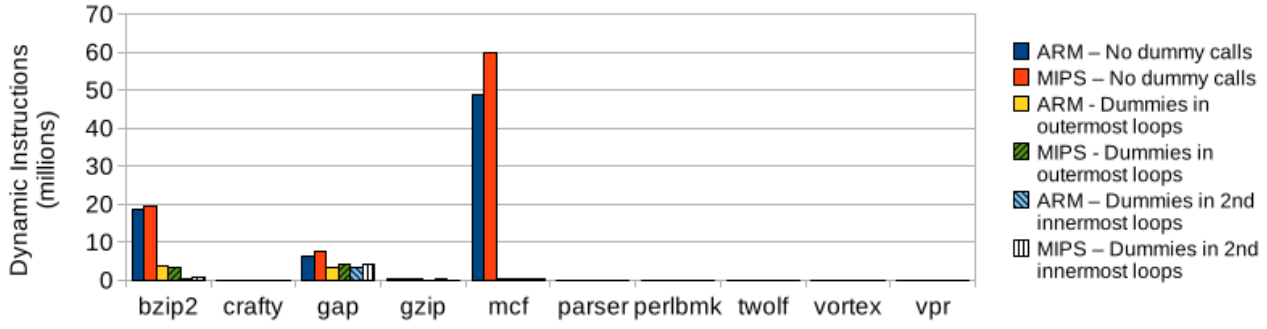
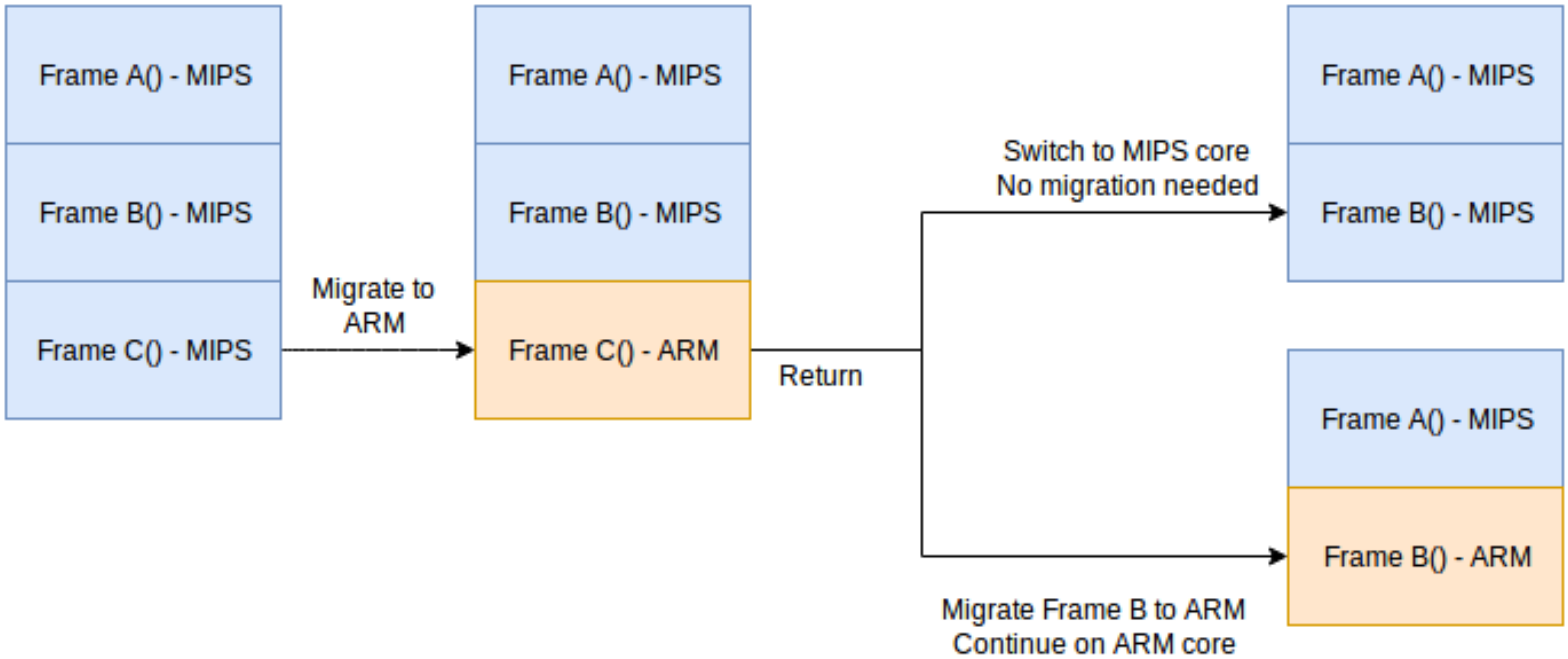


Figure 4. The expected time to the next call, under three situations: no dummy calls have been added, dummy calls to outermost loops have been added, and dummy calls to second-innermost loops have been added.

Single frame transformation

- Only transform the stack frame of current function.
- When returning to previous function, decide whether to migrate to current processor or switch to previous processor.
- Migration cost of only one frame incurred.
- Need ability to migrate inside function (loop migration).



Live variables

- At some execution point in the function, there will be live and dead variables on the stack.
- Optimisation to stack transformer to ignore dead variables can reduce migration cost.
- Compiler will assist in keeping track of live+dead variables.

The End