УДК 004.94

Сравнение производительности библиотек JIT-компиляции для разработки функционального симулятора центрального процессора

Державин А.А.[1], Шурыгин А.А.[1], Шамшура Е.С.[2], Петушков И.В.[1]

[1] ФГАОУ ВО «Московский физико-технический институт» (национальный исследовательский университет), 141701, г. Долгопрудный Московской области, Институтский пер., д.9.
[2] ФГАОУ ВО «Национальный исследовательский университет ИТМО», 197101, г. Санкт-Петербург, Кронверкский проспект, д.49, литер А.

Аннотация. Динамическая двоичная трансляция (ДДТ) является ключевой технологией для высокопроизводительных симуляторов центрального процессора (ЦП). В данной работе представлено практическое сравнение производительности современных JIT-библиотек в контексте реализации ДДТ для симуляции процессора RV32I. В рамках исследования был реализован гибридный симулятор, сочетающий интерпретатор для первичного анализа и JIT-транслятор для оптимизации часто исполняемых участков. На основе результатов алгоритмических и синтетических тестов, а также опыта интеграции библиотек, мы формулируем рекомендации по выбору и практическому использованию JIT-инструментов для построения эффективного симулятора.

Введение. Симуляция центрального процессора (ЦП) играет ключевую роль в проектировании, тестировании и оптимизации процессоров, а также в разработке программного обеспечения. Одним из базовых подходов симуляции ЦП является интерпретация, при которой каждая инструкция гостевой системы исполняется последовательно на хозяйской платформе. Эффективной альтернативой интерпретации является техника двоичной трансляции (ДТ [1]). Для целей симуляции полной системы обычно используют именно динамическую двоичную трансляцию (ДДТ). Её основная идея заключается в преобразовании блоков гостевого машинного кода в семантически эквивалентные последовательности инструкций хозяйской архитектуры “на лету”. Кэшируя трансляции “горячих” участков кода и оптимально чередуя фазы трансляции и исполнения кода, можно значительно повысить производительность модели процессора. Этот подход, известный также как JIT-компиляция (Just-In-Time) [2].

В настоящее время существует множество симуляторов ЦП, использующих технику динамической двоичной трансляции, в разной степени близких к промышленной разработке. QEMU (Quick Emulator) [3] является открытым высокопроизводительным эмулятором полной системы. Для ДТ в QEMU используется собственный JIT-компилятор TCG (Tiny Code Generator). Наиболее приближенным к научному сообществу является Pydgin [4] — это исследовательский инструмент, который использует мета-трассирующий JIT-компилятор на основе RPython для автоматической генерации высокопроизводительных симуляторов на основе ДДТ из собственного языка описания архитектуры.

Несмотря на имеющуюся потребность, в научной литературе отсутствуют актуальные исследования, проводящие широкий сравнительный анализ современных JIT-библиотек именно для целей симуляции ЦП с технологией ДДТ. Для восполнения этого пробела мы создали выборку из пяти современных библиотек: LLVM JIT [5], AsmJit [6], Xbyak [7], GNU Lightning [8], MIR [9]. В качестве объекта исследования выбрана популярная открытая RISC-архитектура RISC-V[10]. Таким образом, наш вклад [11] включает создание модульного симулятора с перемежающимися режимами интерпретации и ДДТ, реализацию динамической трансляции поверх нескольких популярных JIT-библиотек, а также последующий сравнительный анализ

производительности и удобства использования этих инструментов в современном C++ окружении.

Основная часть. Разработанная в рамках исследования модель ЦП имеет модульную архитектуру. Целью проектирования было обеспечение быстрого и гибкого процесса прототипирования различных подходов моделирования процессора с использованием техник ДДТ.

Ключевой частью движка трансляции является общая логика — адаптивный алгоритм ДДТ, определяющий режим исполнения: интерпретация, трансляция. Алгоритм основан на механизме подсчета количества исполнений базовых блоков. Решение о трансляции принимается достижением порогового значения для счетчика выполнений. Данная логика динамической двоичной трансляции — адаптированный алгоритм из исследования, в котором детально описаны методы оптимизации высокоскоростной симуляции ЦП [2].

Набор использованных тестов производительности включает в себя как алгоритмические задачи (сортировки, рекурсивные вычисления), так и синтетические тесты (умножение). Основу тестирования составили стандартизированные бенчмарки из открытого репозитория riscv-tests [12], которые были дополнены рекурсивным вычислением чисел Фибоначчи (fibrec). Все тесты были скомпилированы под архитектуру RV32I без стандартной библиотеки.

Анализ результатов выявил существенные различия в эффективности различных методов. Наибольшую производительность в большинстве тестов демонстрирует “multiply”, что объясняется особенностями его реализации, так как алгоритм последовательного сложения и сдвигов хорошо поддается оптимизации JIT-компиляторов. В противоположность – тест “towers”, стабильно показывающий наихудшую скорость из-за глубокой рекурсии и интенсивных операций с указателями.

Выводы. Проведенное исследование демонстрирует, что выбор конкретной JIT-библиотеки оказывает критическое влияние на производительность симулятора, основанного на ДДТ. Полученные результаты формируют практически важную основу для выбора инструментария. Наибольшую производительность в рассмотренных тестах продемонстрировала библиотека MIR, что делает её предпочтительным выбором для задач, требующих максимальной скорости симуляции. Однако стоит отметить, что MIR предлагает низкоуровневый пользовательский C-интерфейс и обладает ограниченной системой сборки, что осложняет ее интеграцию в современные C++ проекты. LLVM JIT показал свою неэффективность для короткоживущих блоков кода, характерных для ДДТ, из-за высоких накладных расходов. Низкоуровневые библиотеки (Xbyak, AsmJit) предлагают высокую производительность и полный контроль, но требуют глубоких знаний ассемблера хозяйской архитектуры. Несмотря на приемлемую производительность GNU Lightning, её система сборки, построенная на комбинации инструментов Autoconf и Make, создает существенные препятствия для интеграции в современные C++ проекты, использующие систему сборки CMake. Кроме того, разработка логики JIT-трансляции осложняется архитектурно-независимым промежуточным представлением, которое напоминает минимальное подмножество всевозможных систем команд и ограничивает выразительность. Таким образом, данное исследование предоставляет сравнительный анализ современных библиотек JIT-компиляции для задачи ДДТ при симуляции ЦП, учитывающий как производительность, так и аспекты практической интеграции.

Список литературы.

1. John Aycock. 2003. A brief history of just-in-time. ACM Comput. Surv. 35, 2 (June 2003), 97–113. https://doi.org/10.1145/857076.857077.

2. Daniel Jones and Nigel Topham. 2008. High Speed CPU Simulation Using LTU Dynamic Binary Translation. In Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers (HiPEAC '09). Springer-Verlag, Berlin, Heidelberg, 50–64. https://doi.org/10.1007/978-3-540-92990-1_6.
3. Bellard F. QEMU, a fast and portable dynamic translator // USENIX Annual Technical Conference, FREENIX Track. 2005. Т. 41. № 46. С. 10–55.
4. Lockhart D., Ilbeyi B., Batten C. Pydgin: Generating fast instruction set simulators from simple architecture descriptions with meta-tracing JIT compilers // 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2015. С. 256–267.
5. Lattner C., Adve V. LLVM: A compilation framework for lifelong program analysis and transformation // International Symposium on Code Generation and Optimization. 2004.
6. asmjit. asmjit // GitHub. URL: https://github.com/asmjit/asmjit (дата обращения: 31.10.2025).
7. herumi. xbyak // GitHub. URL: https://github.com/herumi/xbyak (дата обращения: 31.10.2025).
8. GNU Project. GNU Lightning: a portable dynamic code generation system. GNU Press, 2023. URL: https://www.gnu.org/software/lightning/manual/lightning.html (дата обращения: 30.10.2025).
9. vnmakarov. mir // GitHub. URL: https://github.com/vnmakarov/mir (дата обращения: 31.10.2025).
10. Cui E., Li T., Wei Q. Risc-v instruction set architecture extensions: A survey // IEEE Access. 2023. Т. 11. С. 24696--24711.
11. ProteusLab. JitResearch // GitHub. URL: https://github.com/ProteusLab/JitResearch (дата обращения: 31.10.2025).
12. riscv-software-src. riscv-tests // URL: https://github.com/riscv-software-src/riscv-tests (дата обращения: 10.11.2025).