DVCOMPUTE++ SIMULATOR ДЛЯ ОТЕЧЕСТВЕННЫХ КОМПЬЮТЕРОВ ЭЛЬБРУС

Д.Э. Сорокин (Йошкар-Ола)

Вызовом нашего времени в области программного обеспечения является необходимость перехода во многих сферах на отечественную программно-аппаратную платформу, такую как Эльбрус «архитектуры E2K». Это актуально и для области имитационного моделирования, в частности, дискретно-событийного моделирования, которое позволяет оценивать поведение сложных динамических систем, таких как системы массового обслуживания. К таким задачам сводятся и модели поведения интегральных микросхем, а умение решать такие задачи также важно для целей импортозамещения.

Описываемая авторская разработка DVCompute++ Simulator[‡] (номер свидетельства 2021660726 Роспатента) призвана послужить данной цели перехода на отечественную программно-аппаратную платформу. Это коллекция программных библиотек на языке C++ для создания и запуска дискретно-событийных моделей.

Программные библиотеки реализованы на основе авторского метода моделирования, который позволяет создавать имитации для разных режимов моделирования: последовательного, двух разных режимов распределенной имитации [1, 2] и режима вложенной имитации. Режим последовательной имитации подразумевает собой обычную имитацию на одном компьютере. Один из режимов распределенной имитации реализует оптимистичный метод деформации времени (англ. Time Warp) [3], а другой – консервативный метод нулевых сообщений [4]. Для распределенной имитации предполагается, что она будет запущена на многоядерной (многопроцессорной) системе или суперкомпьютере. Режим вложенной имитации связан с теорией игр, и он может быть использован для выбора оптимальной стратегии поведения, если рассматривать аналог игры в шахматы, но только где ходы противников меняют не расположение фигур на доске, а меняют состояние некоторой имитационной модели.

Все названные режимы моделирования в работе автора используют единый формализм, который позволяет определять модели систем массового обслуживания в терминах дискретных событий, дискретных процессов, а также блоков транзактов, похожих на язык моделирования GPSS. Также адаптированы приемы реактивного программирования, что позволяют моделировать сигналы, задерживать их через очередь событий, преобразовывать, объединять и разделять, подписывать «слушателей» на их обработку, то есть, можно моделировать аппаратуру, такую как интегральные схемы. Причем все элементы этого формализма взаимосвязаны, что позволяет использовать их вместе в рамках одной имитационной модели. Стоит особо отметить, что это одинаково справедливо для всех названных режимов имитации, включая и вложенную имитацию, и режим оптимистичной распределенной имитации, где постоянно происходят откаты состояния модели в рамках имитации.

Сам метод моделирования един. Режимы имитации отличаются главным образом тем, какую реализацию мы выбираем для очереди событий, через которую все работает, и какую реализацию для изменяемых ссылок, через которые и выражается состояние модели. Метод широко использует приемы функционального программирования, где имитационная модель рассматривается как композиция вычислений.

[‡] URL: https://www.aivikasoft.com/

Описание самого метода можно увидеть в более ранних работах автора, посвященных его симуляторам Айвика [5, 6] и DVCompute Simulator (номер свидетельства 2021660590 Роспатента) [7]. Симуляторы разные и сильно отличаются в деталях. Каждый из них имеет независимую реализацию, учитывающую специфику того языка программирования, на котором их реализовал автор настоящей статьи. Однако все три симулятора, наряду с описываемым DVCompute++ Simulator, реализуют общий единый формализм, одну и ту же математическую идею.

Для примера возьмем фрагмент кода на языке GPSS, где транзакт захватывает прибор, чтобы сымитировать некоторую активность.

Листинг 1

```
PREEMPT PROF,PR,Add,5
ADVANCE (Exponential(1,0,200))
RETURN PROF
```

В ходе обычного выполнения транзакт временно захватывает некоторый прибор PROF, а затем имитирует некоторую активность в течение модельного времени, которое имеет показательное распределение. Однако в случае неожиданного вытеснения у транзакта отбирается прибор, и транзакт передается цепочке блоков, начинающихся с неуказанной здесь метки Add. Вообще говоря, реализация подобного поведения в симуляторе представляет собой определенную сложность.

На языке C++ аналогичное поведение определяется выражением, задающим некоторое вычисление, которое приведено в листинге 2, где для краткости убрана лишь одна спецификация типа.

Листинг 2

```
preempt_block(prof, PreemptBlockMode<Item> {
  true, std::make_optional(add_chain(line, prof)), false
})
.then(advance_block (random_exponential_process_(200.0)))
.then(return_block(prof))
```

Здесь мы видим, что вычисления соединяются с помощью комбинатора then. Так, вычисление preempt_block из C++ аналогично блоку PREEMPT из языка моделирования GPSS. Передаются аналогичные параметры, а в случае вытеснения транзакт будет передан на обработку другому вычислению, которое возвращает функция add_chain. Если читатель помнит, в случае GPSS здесь стояла метка Add.

Вычисление advance_block может имитировать вообще любые задержки, но здесь используется дискретный процесс, который задерживает модельное время на аналогичный интервал, имеющий такое же показательное распределение.

Автор предполагает, что многие имитационные модели могут быть записаны таким способом на языке программирования C++. Есть аналоги и других блоков, например ASSEMBLE, GATHER и SPLIT. Что касается блока SELECT, то его нужно расписывать явно через композицию вычислений.

В то же время запись моделей не исчерпывается аналогами блоков GPSS. Можно определять модели, оперируя дискретными событиями или дискретными процессами.

Листинг 3

```
[=](SharedRef<Transact<Item>>&& transact) {
  return preempt_facility(prof, transact, FacilityPreemptMode<Item> {
    true, std::make_optional(add_process(line, prof)), false
  })
  .then([](Unit&& unit) {
    return random_exponential_process_(200.0);
```

```
  })
  .then([=](Unit&& unit) {
    return return_facility(prof, transact);
  })
  .then([=](Unit&& unit) {
    return pure process(transact);
  });
}
```

Фактически, то, что мы видим в листинге 2, преобразуется симулятором в что-то похожее на код, который приведен в листинге 3. Здесь возникает функция, которая по заданному входному транзакту возвращает дискретный процесс, на выходе которого тоже ожидается транзакт. В рамках дискретного процесса и осуществляется моделирование желаемой активности: здесь и захват прибора с возможным вытеснением, и задержка с показательным распределением, и освобождение прибора, и, наконец, возвращение транзакта как результата работы дискретного процесса.

На самом деле, это еще не конец истории. Дискретный процесс, приведенный в листинге 3, преобразуется в череду вычислений, оперирующих напрямую с очередью событий, но преобразования эти уже сложны и нетривиальны. Саму математическую идею этого преобразования можно увидеть в упомянутых работах автора про Айвику [5, 6]. Важно, что в итоге все моделирующие активности работают через очередь событий.

Возвращаясь к теме поддержки разных режимов имитации, стоит заметить, что приведенные фрагменты кода на C++ одинаково везде будут работать, и в случае последовательной имитации, и даже в случае оптимистичного метода распределенного моделирования, где обычно возникает много откатов во время имитации. Только использованные в примерах функции будут импортированы из разных пространств имен C++. Во всем же остальном функции будут совпадать для разных режимов имитации. Тем более, что кодовая база у всех режимов общая, исходные файлы те же самые.

В настоящее время в DVCompute++ Simulator реализовано самое главное – моделирующее ядро. Фактически, это внутренний предметно-ориентированный язык, на котором в рамках языка C++ можно описывать широкий класс имитационных моделей. Пока нет вывода графиков и записи результатов в файлы и базу данных, но автор уже проделывал такое для Айвики. Поэтому добавление подобной функциональности для DVCompute++ Simulator остается лишь вопросом времени.

Что касается распределенной имитации, то обмен сообщениями между логическими процессами реализован через использование протокола MPI. Существуют готовые реализации обоих модулей распределенного режима.

Для тестирования распределенных модулей автор использовал воспроизводимые генераторы псевдослучайных чисел, которые приводят к одинаковым результатам имитации независимо от задержек сети, откатов в имитации и прочих факторов. Это крайне важная проверка для модулей распределенного моделирования.

Дистрибутив DVCompute++ Simulator распространяется автором по лицензии, которая допускает бесплатное использование для некоммерческих и образовательных проектов. Для режимов последовательной и вложенной имитации полностью доступен исходный код. Для обоих бесплатных модулей распределенной имитации могут действовать ограничения.

Симулятор для всех поддерживаемых режимов имитации был сначала протестирован автором на аппаратной платформе Intel x86-64 с использованием

операционных систем Linux, Windows, macOS, а также компиляторов GCC, Clang и Visual C++.

Автор специально выбрал в качестве языка реализации C++ из-за его хорошей переносимости, а также из-за того, что с ним можно работать без использования интернета. Здесь мы имеем минимальные технологические риски.

Этот выбор в полной мере оправдался при тестировании симулятора на отечественной аппаратно-программной платформе Эльбрус E2K с использованием штатного компилятора LCC для соответствующей системы «Эльбрус Линукс». Автор проверил работу только для открытой части симулятора, что подразумевает собой само моделирующее ядро, а также включает поддержку последовательного и вложенного моделирования. Есть высокие шансы того, что все заработает и для двух режимов распределенного моделирования, то есть, что симулятор можно будет использовать для суперкомпьютеров, построенных на базе Эльбруса.

Стоит признать, что программирование имитационных моделей на языке C++ не всегда является простым занятием. Поэтому автор задумывается и об использовании некоторого промежуточного языка, на котором можно было бы определять части модели, которые бы потом автоматически переводились в код на языке C++. При этом желательно будет сохранить эффективность получаемого кода, как если бы мы сравнивали с кодом, написанным на C++ вручную опытным программистом.

В качестве заключения хочется отметить, что описанных в статье результатов автору удалось добиться за счет использования приемов функционального программирования, что позволило взглянуть на задачи имитационного моделирования с другого ракурса. Это было хобби, увлечение в свободное время. Однако, по мнению автора полученные результаты могут быть полезны для решения сложных задач, которые ранее могли показаться труднорешаемыми. Тем более, это возможно делать с минимальными технологическими рисками, используя российские компьютеры.

Наконец, за предоставленный удаленный доступ к компьютеру Эльбрус с архитектурой E2K автор благодарит компанию МЦСТ, разработчика одноименных процессоров, и Рамиля Саттарова (с его разрешения), который является представителем команды «e2k-dev», энтузиастов-программистов платформы Эльбрус.

Литература

1. **Окольнишников В. В.** Представление времени в имитационном моделировании // Вычислительные технологии. 2005. Т. 10. № 5. С. 57-80.
2. **Fujimoto R. M.** Parallel and distributed simulation systems. Wiley Interscience, 2000. 302 с.
3. **Jefferson D.R., B. Beckman** et al. The Time Warp operating systems // 11th Symposium on Operating Systems Principles. 1987. No. 21. P. 77–93. DOI: 10.1145/37499.37508.
4. **Chandy M., Misra J.** Distributed simulation: A case study in design and verification of distributed programs // IEEE Transactions on Software Engineering SE-5. 1979. No. 5. P. 440–452. DOI: 10.1109/TSE.1979.230182.
5. **Сорокин Д. Э.** Айвика: имитационное моделирование в терминах вычислений // Седьмая всероссийская научно-практическая конференция «Имитационное моделирование. Теория и практика» ИММОД-2015. М., 2015. Т. 2. С. 262-266.
6. **Сорокин Д. Э.** Распределенное имитационное моделирование с Aivika // Прикладная информатика. 2019. Т. 14. № 4 (82). С. 73–89. DOI: 10.24411/1993-8314-2019-10028.
7. **Сорокин Д. Э.** DVCompute Simulator для дискретно-событийного моделирования // Прикладная информатика. 2021. Т. 16. № 3. С. 93–108. DOI: 10.37791/2687-0649-2021-16-3-93-108.