ло на основе экспериментальных данных построить ансамбль искусственных нейронных сетей для прогнозирования динамики восстановления функционального состояния организма после однократной истощающей физической нагрузки.

Полученные результаты могут найти применение для составления графика физических нагрузок для студентов высших учебных заведений, а также при определении допуска спортсмена к последующим повторным попыткам в текущем соревновательном периоде.

1. Баевский, Р.М. Вариабельность сердечного ритма: теоретические аспекты и возможности клинического применения / Р.М. Баевский, Г.Г. Иванов // Новые методы электрокардиографии: под ред. С.В. Грачева, Г.Г. Иванова, А.Л. Сыркам. – М.: Техносфера, 2007. – С. 473–498.

2. Гаврилова, Е.А. Ритмокардиография в спорте / Е.А. Гаврилова. – СПб: Изд-во СЗГМУ, 2014. – 164 с.

3. Прохожий, С.А. Прогнозирование восстановления функционального состояния организма после истощающей физической нагрузки / С.А. Прохожий, Э.С. Питкевич // Веснік Віцебскага дзяржаўнага ўніверсітэта імя П.М. Машэрава. – 2020. – № 1. – С. 16-20 URL: https://rep.vsu.by/handle/123456789/21433 (дата обращения: 30.01.2023).

ОБ ОПТИМИЗАЦИЯХ РАБОТЫ С УЧАСТКАМИ ПАМЯТИ НА ПРИМЕРЕ ПЛАТФОРМЫ .NET И КЛАССА SPAN<T>

М.Г. Семёнов
Витебск, ВГУ имени П.М. Машерова

Современные информационные системы предназначены для работы с большим количеством информации. Типичный процесс обработки запроса подразумевает получение данных из одного или нескольких источников (баз данных, внешних сервисов, кешей и т.д.), фильтрацию, сортировку и агрегацию полученных данных. Как правило для этого используются обобщённые коллекции, копии которых создаются при каждой новой операции. При этом размеры выборки, размеры самих объектов моделей данных, а также количество различных промежуточных обработчиков зачастую достаточно велико. В следствии чего, память сервера достаточно быстро занимается короткоживущими объектами, что вызывает увеличение количества сборок мусора и, как следствие, уменьшение производительности информационной системы.

Аналогичная ситуация возникает при работе с текстовой информацией. Обработчики строк зачастую создают большое количество копий и как следствие происходит заполнение памяти множеством вспомогательных строк. Существуют подходы (например, применение StringBuilder), которые могут решать данную проблему, но только в частных случаях.

Для увеличения производительности в общем случае можно применять классические подходы. А именно, можно использовать небезопасные блоки кода и указатели для прямого манипулирования памятью. Однако, этот подход сопряжен со значительными рисками. Манипуляции с указателями подвержены ошибкам, таким как переполнение, доступ к нулевому

указатель и др. Если баг затрагивает только стек или области статической памяти, то он не всегда слишком опасен, но, если он затронет критические области системной памяти, это может привести к сбою всего приложения.

Целью настоящей работы является анализ безопасных способов работы с непрерывными областями памяти на примере платформы .NET и класса System.Span<T>.

Материал и методы. Материалом для исследования являются документация ASP.NET Core и полученные ранее опыт в данном направлении. Методы исследования: анализ источников, изучение и обобщение сведений, сравнительный анализ.

Результаты и их обсуждение. Span<T> (пространства имен System) [1] – это новый значимый тип платформы .NET. Он позволяет представлять непрерывные области памяти, независимо от того, связана ли эта память с управляемым объектом, предоставляется собственным кодом через взаимодействие (interop) или находится в стеке. При этом, сохраняется безопасный доступ с характеристиками производительности уровня классических массивов. Данный тип предназначен для использования в ситуациях, когда необходимо обрабатывать большие объемы данных и/или избегать дополнительного выделения памяти, но при этом необходимо оптимизировать производительность.

Приведем примеры типовых ситуаций, в которых можно эффективно применять Span<T>:

1) Работа с массивами данных и их частями;

2) Работа со строками с большим количеством обращений к подстрокам;

3) Буферы в памяти в «неуправляемом» (unmanaged) коде.

Например, при анализе большого текста, зачастую необходимо разбить его на меньшие части с применением метода substring. Для каждой из таких частей будет создан новый объект, который будет дублировать информацию в уже имеющемся. Далее, такая ситуация может повториться, и некоторые подстроки будут разбиты на еще меньшие для дальнейшей обработки, что приведет к созданию еще большего количества копий. Данный подход уменьшает как производительность с точки зрения процессорного времени (копирование элементов), так и с точки зрения памяти (большое количество дублирований, дополнительные сборки мусора).

Альтернативным подходом будет применение класса Span<T> и его метода метод Slice, который сформирует «срез» текущей строки, начиная с указанного индекса, для указанной длины. В такой ситуации «новая» переменная будет ссылаться на те же самые элементы, но с точки зрения исходного кода, это будет отдельная самостоятельная сущность, которую можно передавать в другие методы и обрабатывать. Таким образом мы избегаем дополнительного выделения памяти и последующих дополнительных вызовов дорогостоящего (по времени) процесса сборки мусора. При-

меняя данный подход, мы значительно увеличим производительность, при этом код будет написан в современном безопасном стиле.

Был приведен один из типовых примеров использования Span<T>. Для получения дополнительной информации о способах и примерах использования Span<T> смотрите [1, 2].

Заключение. Использование языков программирования и платформ с системами автоматического управления памятью и сборкой мусора при разработке современных приложений значительно уменьшило количество ошибок и положительно повлияло на разработку программного обеспечения в целом. Однако, еще остается ряд ситуаций, в которых для оптимизации производительности приходится писать «небезопасный» код с использованием указателей. Предложенный компанией Microsoft подход с применением класса System.Span<T> позволяет использовать современные безопасные подходы для работы с непрерывными областями памяти, значительно увеличивая производительность приложения.

1. Toub, S. All About Span: Exploring a New .NET Mainstay / S. Toub // MSDN Magazine. – 2018. – V. 33, N. 1.
2. Writing High-Performance Code Using Span<T> and Memory<T> in C# [Электронный ресурс]. – Режим доступа: https://www.codemag.com/Article/2207031/Writing-High-Performance-Code-Using-SpanT-and-MemoryT-in-C. – Дата доступа: 30.01.2023.

ОБ ОТОБРАЖЕНИИ СТРОКОВЫХ ЛИТЕРАЛОВ В ПРОСТРАНСТВО ТИПОВ ЯЗЫКА ПРОГРАММИРОВАНИЯ СИ++11

С.В. Сергеенко
Витебск, ВГУ имени П.М. Машерова

Обработка текстовых данных является одной из основных составляющих информационных технологий. Вопрос эффективной программной реализации такой обработки тесно связан с представлением и обработкой констант на этапе трансляции. Язык программирования Си++ позволяет описывать обработку констант с помощью шаблонов и constexpr функций. Однако строковые литералы не могу быть параметрами шаблонов и результатами функций [1].

Цель работы – получить представление строковых данных (констант) программы, доступное компилятору на этапе трансляции, средствами языка программирования Си++ стандарта 2011 года.

Материал и методы. Материалом исследования являются способы описания в исходном коде программы преобразований строковых литералов на этапе трансляции. Поставленная цель достигается средствами обобщенного программирования по средствам шаблонов в языке программирования Си++ с учетом возможностей стандарта 2011 года.