ВИКОРИСТАННЯ «НУЛЬОВОГО» ЗБИРАЧА СМІТТЯ ДЛЯ ПРИСКОРЕННЯ ОБЧИСЛЕНЬ ЗАДАЧ ОБРОБЛЕННЯ ПРИРОДНОЇ МОВИ НА ПЛАТФОРМІ .NET

Юсин Яків Олексійович
ORCID ID: http://orcid.org/0000-0001-6971-3808
асистент кафедри ПЗКС
Національний технічний університет України «КПІ ім. Ігоря Сікорського», Україна

На сьогодні мови програмування використовуються в багатьох галузях практичної діяльності людини – в тому числі і для наукових обчислень. При цьому існує велика різниця між загальною популярністю мов програмування та їх популярністю для використання в наукових задачах. Таку різницю можливо пояснити тим, що при виборі мови програмування для обчислень важливими є такі критерії, як швидкодія, масштабованість, гнучкість, легкість використання. Досить популярними для наукових обчислень є мови програмування зі збирачами сміття (garbage collector – GC), які впливають на їх швидкодію. Написання коду з високою швидкодією на таких мовах (включаючи мови платформи .NET) потребує або слідування спеціальним шаблонам, або підключення окремих бібліотек, написаних на інших мовах без збирача сміття.

Проте, з випуску .NET Core 2.0, стало можливим підключення власних збирачів сміття до платформи .NET [1]. Це дає змогу використовувати так званий «нульовий» збирач сміття – який ніколи не виконує збирання сміття – для прискорення наукових обчислень на платформі .NET (за рахунок збільшеного використання пам'яті), в тому числі, для задач оброблення природної мови.

Таким чином, метою даної роботи є прискорення обчислень задач оброблення природної мови на платформі .NET, шляхом використання «нульового» збирача сміття. Об'єктом даного дослідження є процес виконання обчислень на платформі .NET, предметом – «нульовий» збирач сміття для платформи .NET.

Для реалізації власного збирача сміття для платформи .NET, необхідно реалізувати декілька стандартних інтерфейсів CoreCLR: IGCHandleManager, IGCHandleStore та IGCHeap [2]. Також модуль збирача сміття повинен оголосити дві зовнішні функції – GC_Initialize, яка відповідає за ініціалізацію, та GC_VersionInfo, яка відповідає за повернення підтримуваної версії CoreCLR. Якщо підтримувана версія є меншою, ніж поточна версія CoreCLR, буде завантажено збирач за замовчуванням.

IGCHeap – основний інтерфейс для реалізації власного збирача сміття, який описує основну функціональність збирання сміття. Більшість методів цього інтерфейсу, у випадку «нульового» збирача, можливо реалізувати заглушками. Основними методами, що мають реалізацію у випадку такого збирача, є метод Initialize, а також три версії методу Alloc, який власне виділяє пам'ять для об'єктів. Відповідно, метод Alloc для створення об'єктів використовує блок наперед виділеної пам'яті, для якого зберігається вказівник на ще не використану область. В випадку, якщо ця пам'ять вже повністю використана, виділяється новий блок і вказівник переміщується на нього.

Для того, щоб вказати назву файлу з реалізацією збирача сміття, використовується змінна середовища COMPlus_GCName (при цьому бібліотека повинна знаходитись в одному місці з застосунком). В випадку, якщо ця змінна

вказана при старті застосунку, CoreCLR завантажує цю бібліотеку, перевіряє версію за допомогою виклику функції GC_VersionInfo, та виконує ініціалізацію збирача сміття за допомогою виклику функції GC_Initialize.

Для тестування швидкодії та використання пам'яті при використанні «нульового» збирача сміття, реалізовано чотири різних тестових програми, дві з яких реалізовують синтетичні сценарії, і дві – справжні робочі сценарії (обчислення різних функцій з галузі оброблення природної мови). Запуск відбувався на робочій машині з 16 Гб оперативної пам'яті та процесором Intel Core i7-4790 @ 3.6 GHz.

Для першого синтетичного сценарію – розміщення в пам'яті великої кількості рядків – отримано прискорення більше, ніж в 4 рази, за рахунок збільшення використання пам'яті на два порядки.

Для другого синтетичного сценарію – виконання пакування структури – отримано прискорення близько в 1.5 рази за рахунок подвоєного-потроєного використання пам'яті.

Для першого робочого сценарію – обчислення відстані Левенштейна - виконання обчислення уповільнилось, з близько 34 секунд до 37 секунд. При цьому використання пам'яті збільшилось приблизно на 2 порядки. Це може бути пов'язано з використанням підходу динамічного програмування для реалізації обчислення відстані Левенштейна.

Для другого робочого сценарію – обчислення матриці кореляції термів – отримане прискорення становить близько 4-14%. При цьому збільшення використання пам'яті в абсолютних числах становить лише близько 50 Мб (у відносних – в три рази), що на сьогодні є невеликою різницею. В випадку, якщо таке обчислення потрібно запускати часто (наприклад, при надходженні нових текстів, якщо це потоковий аналіз), навіть мінімальне прискорення в 4% буде відчутним результатом.

Висновки. В даній роботі розглянуто метод прискорення обчислень задач оброблення природної мови на платформі .NET за допомогою використання «нульового» збирача сміття. Тестування на чотирьох різних сценаріях (два з яких синтетичні та симулюють різне навантаження на збирач сміття, та два робочих) показало, що головна мета даної роботи досягнута – на трьох сценаріях (два синтетичних та один робочий) отримано прискорення виконання обчислень. На фізичній машині з 16 Гб оперативної пам'яті та процесором Intel Core i7-4790 @ 3.6 GHz отримане прискорення знаходиться в межах від 4 до 425 відсотків в залежності від сценарію. При цьому, при найгіршому сценарії, відбувається збільшення використання пам'яті на два порядки.

Таким чином, ефективне використання «нульового» збирача сміття можливе при дотриманні наступних умов:

1. встановлена достатня кількість оперативної пам'яті для того, щоб уникнути її переповнення та запису на диск;
2. програмний код створює велике навантаження на збирач сміття, наприклад, створюючи велику кількість короткоживучих різноманітних об'єктів.

Список використаних джерел:

1. Gillespie, S. (2017). Standalone GC Loader Design. Вилучено з GitHub: https://github.com/dotnet/coreclr/blob/master/Documentation/design-docs/standalone-gc-loading.md.
2. Kokosa, K. (2018). Custom GC. In K. Kokosa, Pro .NET Memory Management (с. 1042-1045). New York: Apress.