С.В. Болтанов

ПРОЕКТИРОВАНИЕ И ИССЛЕДОВАНИЕ ТИПОВОГО РЕШЕНИЯ «МЕТОД ПРЕМЕННЫХ НАПРАВЛЕНИЙ»: АЛГОРИТМЫ БАРЬЕРНОЙ СИНХРОНИЗАЦИИ

(Самарский государственный аэрокосмический университет)

В статье продолжаются исследования типового решения «метод переменных направлений» и его эффективность, рассмотренные в [1] и [2].

При использовании типового решения «метод переменных направлений» возникла необходимость реализации и исследовании эффективности барьерной синхронизации и асинхронного ввода-вывода информации.

Синхронизация — это механизм, позволяющий наложить ограничения на порядок выполнения потоков. Путем синхронизации регулируется относительный порядок выполнения потоков и решается любой конфликт между потоками, который мог бы привести к нежелательному поведению программы. Синхронизация обеспечивает координацию выполнения потоков и управление совместно используемыми данными.

Основным свойством большинства параллельных итерационных алгоритмов является зависимость результатов каждой итерации от результатов предыдущей.

Причина, по которой в работе выбран данный способ синхронизации – барьерная синхронизация – в том, что, например, обычный мьютекс – это механизм ядра, а значит, для его работы требуется переход из режима пользователя в режим ядра. Это дорогая операция, но она предоставляет мощный механизм синхронизации, который можно использовать через границы процессов. Однако во многих приложениях синхронизация необходима внутри единственного процесса. Поэтому способность мьютекса работать через границы процессов не нужна и лишь приводит к непроизводительным затратам.

В работе рассматривается три алгоритма барьерной синхронизации: разделяемого счетчика, флагов и управляющего потока и симметричных барьеров.

Самый простой способ реализовать барьер – использовать разделяемую целочисленную переменную-счетчик count с нулевым начальным значением. Пусть есть n рабочих процессов, которые должны собраться у барьера после решения своих задач. Как только процесс дойдет до барьера, он увеличит значение переменной-счетчика count. Когда значение переменной-счетчика count станет равным n, все процессы смогут продолжить работу. Пример реализации следующий:

```
int count = 0;                                          (1)
process Worker[i=1 to n] {
    Код реализации задачи i;
```

```
    <count = count + 1;>
    <await (count == n);>
}
```

Недостаток метода заключается в возможности конфликтов обращения к памяти и необходимости обновлять кэш. Однако этой проблемы можно избежать, если на машине есть неделимые инструкции увеличения и согласованная кэш-память.

Второй из способов реализовать барьерную синхронизацию – использовать переменные-флаги. Пусть есть массив целых arrive[1:n] с нулевыми начальными значениями. Заменим операцию увеличения счетчика count в программе (1) присваиванием arrive[i] = 1. Тогда глобальным инвариантом станет предикат

```
count == (arrive[1] + … + arrive[n]).
```

Если элементы массива arrive хранится в разных строках кэш-памяти, то конфликтов обращения к памяти не будет.

Обе проблемы – конфликт обращения к памяти и обнуления массива – можно решить, используя дополнительный набор разделяемых значений и еще один процесс-координатор, Coordinator. Пусть каждый процесс Worker вместо того, чтобы суммировать элементы массива arrive, ждет пока не станет истинным единственно логическое значение. Пусть continue[1:n] – дополнительный массив целых с нулевыми значениями. После того как Worker[i] присвоит 1 элементу arrive[i], он должен ждать, пока значением переменной continue[i] не станет 1.

```
arrive[i] = 1;                                          (2)
<await(continue[i] == 1);>
```

Процесс Coordinator ожидает, пока все элементы массива arrive не станут равны 1, затем присваивает значение 1 всем элементам массива continue.

```
for [i = 1 to n] <await (arrive[i] == 1);>      (3)
for [i = 1 to n] continue[i] = 1;
```

Пусть каждый процесс при достижении им барьера устанавливает собственный шаг. Если W[i] и W[j] – два процесса, то симметричный барьер для них реализуется следующим образом.

```
/*код барьера для рабочего процесса W[i] */
    <await (arrive[i]) == 0);>
    arrive[i] = 1;
     <await (arrive[j] == 1);>
    arrive[j] = 0;
```

```
/*код барьера для рабочего процесса W[j] */
    <await (arrive[j]) == 0);>
    arrive[j] = 1;
    <await (arrive[i] == 1);>
     arrive[i] = 0;
```

Первая строка в алгоритме необходима, чтобы не допустить ситуации, когда процесс успеет вернуться к барьеру и установить собственный флаг до того, как другой процесс сбросит флаг на предыдущем использовании барьера.

Третий метод барьерной синхронизации – с помощью симметричных барьеров. При программировании численных методов процессы выполняют одинаковую последовательность действий. Если эти процессы еще и выполняются на отдельных процессорах, то достичь барьер они могу одновременно. Симметричные барьеры для n процессов строятся из пар простых двухпроцессных барьеров. Пусть каждый процесс при достижении им барьера устанавливает собственный шаг. Если W[i] и W[j] – два процесса, то симметричный барьер для них реализуется следующим образом.

```
/*код барьера для рабочего процесса W[i] */
    <await (arrive[i]) == 0);>
    arrive[i] = 1;
     <await (arrive[j] == 1);>
     arrive[j] = 0;

/*код барьера для рабочего процесса W[j] */
    <await (arrive[j]) == 0);>
    arrive[j] = 1;
     <await (arrive[i] == 1);>
    arrive[i] = 0;
```

Первая строка в алгоритме необходима, чтобы не допустить ситуации, когда процесс успеет вернуться к барьеру и установить собственный флаг до того, как другой процесс сбросит флаг на предыдущем использовании барьера.

В рамках выпускной квалификационной работы магистра описанные выше методы используются для программной реализации типового решения «метод переменных направлений» для барьерной синхронизации потоков при вычислениях в численном методе. Реализация барьеров с использованием разделяемых переменных позволяет обеспечить переносимость и высокоэффективное исполнение кода типового решения на программных платформах Win32 и Posix.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Болтанов С.В.Анализ эффективности численного моделирования с использованием типового решения «Метод переменных направлений» // Перспективные информационные технологии в научных исследованиях,

проектировании и обучении (ПИТ 2012): труды научно-технической конференции с международным участием и элементами научной школы для молодежи, посвященной 40-летию кафедры информационных систем и технологий СГАУ / под ред. С.А. Прохорова. – Самара: Издательство Самарского научного центра РАН, 2012. – с.67-70

2. Востокин С.В., Литвинов В.Г., Макагонова Д.Д., Хайрутдинов А.Р.Представление алгоритмов высокопроизводительных вычислений в нотации TEMPLET // Перспективные информационные технологии в научных исследованиях, проектировании и обучении (ПИТ 2012): труды научно-технической конференции с международным участием и элементами научной школы для молодежи, посвященной 40-летию кафедры информационных систем и технологий СГАУ / под ред. С.А. Прохорова. – Самара: Издательство Самарского научного центра РАН, 2012. – с.74-76
3. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. : Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 512 с.: ил. – Парал. тит. англ.