ОРГАНИЗАЦИЯ УПРЕЖДАЮЩЕГО КЭШИРОВАНИЯ ДЛЯ ПРИЛОЖЕНИЙ, ИСПОЛНЯЮЩИХСЯ В СЕТЕВОЙ СРЕДЕ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

САХАРОВ И.Е., асп.

Представлен вариант организации метода упреждающего кэширования данных на основе метода разделения потоков. Также представлены особенности функционирования метода упреждающего кэширования и системная архитектура модуля упреждающего кэширования.

In this article author suggests prefetch caching method in terms of separation threads method. Also we can see some working features of prefetch caching method and system architecture of prefetch caching module.

Ключевые слова: распределенные вычисления, кэширование, упреждающее кэширование, разделенные потоки, повышение производительности.

Key words: distributed computing, caching, prefetch caching, separation threads, increase in productivity.

Введение. Одной из современных тенденций развития высокопроизводительных систем является интеграция различных ресурсов в единую сетевую среду распределенных вычислений (ССРВ) [1]. Данная среда позволяет использовать разнородные средства как единый вычислительный ресурс. ССРВ состоит из множества территориально размещенных вычислительных систем, соединенных различными каналами связи, характеристики которых могут динамически изменяться в широких пределах, и представляет собой GRID-систему.

Приложения (задачи) пользователей выполняются на одной или нескольких вычислительных систем, входящих в состав ССРВ. При запуске задачи на выполнение могут возникнуть проблемы, связанные с тем, что исходные данные, необходимые для счета, располагаются в других компонентах ССРВ. Из-за большого территориального расположения компонент ССРВ возникает задержка передачи данных по каналам связи и задержка обращения к дисковым накопителям.

Разница между временем вычисления одной процессорной операции и временем доступа к единице дисковой памяти постоянно увеличивается. Эта тенденция служит причиной того, что приложения должны осуществлять дополнительную выборку данных из дисковой памяти до момента обращения к этим данным. Основной преградой для получения выигрыша от этой техники на практике является сложность получения точных и своевременных предвыборок. Предлагается использовать новый метод упреждающего кэширования для автоматической генерации точных и своевременных предвыборок без участия программиста.

Понятие упреждающего кэширования. Упреждающее кэширование (УК) представляет собой процесс независимого *заблаговременного* копирования данных из одной памяти в другую в течение всего процесса выполнения приложения.

Существующие методы доступа к данным, основанные на примерах, истории и статистическом анализе, не позволяют решить задачу эффективного доступа к данным. Предлагаемый метод упреждающего кэширования, основанный *на методе разделения потоков*, позволяет решить эту задачу для большого множества различный приложений с интенсивным обменом данными. Основная идея заключается в разложении приложения на два потока: вычислительный поток, содержащий неизмененный код оригинальной программы, включающий все вычислительные операции и все операции ввода/вывода, и упреждающий поток, содержащий все оригинальные инструкции, которые имеют отношение к вводу/выводу. Упреждающий поток должен исполняться быстрее вычислительного потока.

Архитектура упреждающего кэширования. Все современные разработки принято делить на три направления:

1. Предсказуемое упреждающее кэширование. Самые ранние разработки систем упреждающего кэширования базировались на методах, основанных на примерах и истории обращений [2].

2. Специальные приложения, контролирующие упреждающие выборки и кэширование. Подобными системами занимался Patterson [3].

3. Специальное компилирование приложений. Автоматическое компилирование приложения для использования упреждающего кэширования. Подобными системами занимались T. Mowry [4], а также Chang и Gibson [5], которые разработали средство SpecHint.

Предлагаемый метод упреждающего кэширования на основе разделения потоков относится к третьему направлению.

Системная архитектура. Упреждающее кэширование основывается на четырех компонентах: транслятор исходного кода, библиотека упреждающего кэширования, загрузочный модуль и модуль упреждающего кэширования (рис. 1).

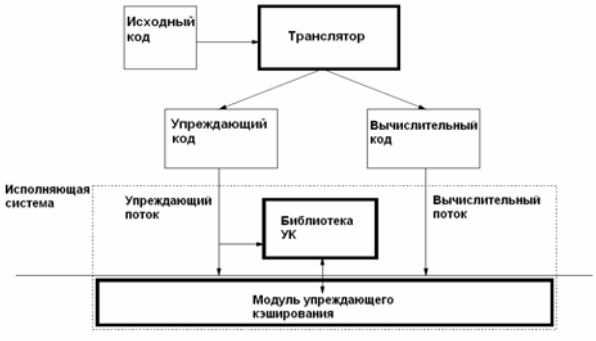

Рис. 1. Системная архитектура

Транслятор кода создает упреждающий поток из приложения путем выборки тех частей

кода, которые относятся к вводу/выводу и урезанию всех вычислительных частей. Причем все вызовы ввода/вывода заменяются вызовами из библиотеки упреждающего кэширования. Само исходное приложение формирует вычислительный поток. Имеется однозначное соответствие между упреждающими вызовами в упреждающем потоке и вызовами ввода/вывода из оригинального приложения. Каждый упреждающий вызов обслуживается загрузочным модулем. Каждому упреждающему вызову ставится в соответствие логический адрес требуемого блока данных. Множество упреждающих вызовов заносится в пользовательскую очередь упреждающих вызовов. Когда пользовательская очередь упреждающих вызовов заполняется, модуль упреждающего кэширования осуществляет системный вызов, который передает эту очередь на выполнение модулю упреждающего кэширования.

Синхронизация строится на идентификационных номерах операций ввода/вывода и позволяет избежать отставания упреждающего потока от вычислительного потока.

Упреждающий поток. Основными операциями модуля упреждающего кэширования являются (см. рис. 2, 3): create_prefetch_thread (prefetch_function) – функция создания упреждающего потока; prefetch_xxx() – это множество функций, которые заменяют в упреждающем потоке все стандартные функции ввода/вывода; inform_open (file_pointer) и inform_close (file_pointer) – функции, необходимые для информирования упреждающего потока об открытии или закрытии файлов вычислительным потоком; synchronize(synchronization_point, type) – это функция синхронизации двух потоков; send_fileptp (file_pointer), receive_fileptr (file_pointer) – функции пересылки файловых указателей между потоками; send_xxx() и receive_xxx() – функции передачи информации между потоками (см. рис. 3, символы XX указывают данные функции).

Процесс генерации упреждающего потока из вычислительного потока лежит в основе транслятора. Алгоритмы внутреннего и внешнего анализа функций представляют собой общеизвестные алгоритмы статического анализа кода программы.

Алгоритм может включать много лишних выражений, которые никогда не заканчиваются операциями ввода/вывода или не влияют на операции ввода/вывода. Существующие компиляторы способны осуществлять проверку и убирать лишние части кода, которые не влияют на функционирования потоков.

Анализатор кода включает упрощенную поддержку многопотоковых приложений, написанных под спецификации MPICH.

```
      int fp;

      Вычислительный поток(ВП)

      void main(void){
       int i; int data[100];
##C1   /*создание упреждающего потока*/

 C2   create_prefetch_thread(
##C3   (void*)prefetch_function);
       /*открытие файла данных*/
 C4
      fp=open("mydata.dat"
 C5   , O_RDONLY);
 C6    /*сигнал упреждающему потоку*/
 C7    synchronize(1, signal);


 C8


       for(i=99; i>=0; i--){
         /*ввод-вывод*/
         lseek(fp, i*4, SEEK_SET);
         read(fp, &data[99-i], 4);
         /*вычисления*/
```

```
      Упреждающий поток(УП)

      Void prefetch_fuction(void){
       int I;

$$P1   /*ожидание открытия файла*/
       synchronize(1, wait);
$$P2   /*указание об открытии файла*/
       inform_open(fp);
 P3    /*предвыборка*/
       for(i=99; i>=0; i--){

 P4     /*только ввод-вывод*/
 P5     prefetch_lseek(fp,i*4,SEEK_SET);
        prefetch_read(fp, 4);


$$P6   }
       /*указание о закрытие файла*/
       inform_close(fp);
      }
```

Рис. 2. Пример вычислительного и упреждающего потоков (внутри функции main строчки без ## являются оригинальным кодом приложения, строчки без символа $$ были взяты с оригинального кода приложения)

```
      Вычислительный поток(ВП)**

      void main(void){
       int fp; int data, int index;
 XX    /* FILE *config_fp */
##C1     create_prefetch_thread(
            (void*)prefetch_function);
 C2    fp=open("mydata.dat", O_RDONLY);
       /*посылка дескпритора для УП*/
##C3   send_fileptr(fp);

XXC4   /*config_fp=fopen("config.dat","r");*/

XXC5   /*flseek(config_fp,10,SEEK_SET);*/


XXC6  /*fscanf(config_fp, "%d", &index);*/
       /*ожидание данных от УП*/
##C7   receive_fscanf("%d", &index);
 C8    lseek(fp, index*4, SEEK_SET);
 C9    read(fp, &data, 4);
 C10   data = data + 10;

 C11   close(fp);
      }
```

```
      Упреждающий поток(УП)

      Void prefetch_fuction(void){
       int fp, int index;
       FILE *config_fp;


$$P1   /*получение дескриптора*/
$$P2   receive_fileptr(&fp);
 P3    inform_open(fp);
       config_fp = fopen("config.dat", "r");
 P4    /*ввод-вывод*/
       flseek(config_fp,10,SEEK_SET);

 P5     /*чтение и посылка ВП*/
       send_fscanf(config_fp, "%d", &index);


 P6
 P7     prefetch_lseek(fp,index*4,SEEK_SET);
       prefetch_read(fp, 4);


$$P8
       inform_close(fp);
      }
```

Рис. 3. Обмен данными между потоками (строки, помеченные XX, были изъяты из вычислительного потока и помещены в упреждающий поток, строки, не помеченные $$, были взяты из оригинального кода приложения)

Архитектура исполняющей системы. Структура очереди команд на упреждение имеет следующий вид: {идентификатор файла, номер блока, идентификатор вызова}. Идентификатор файла и блока определяют реальный физический блок данных, который необходимо предвыбрать. Идентификатор вызова определяет вызов,

который инициировал запуск механизма упреждения для данного блока.

Для каждого вызова ставится в соответствие идентификатор этого вызова (рис. 4). Далее ID вызова помещается в очередь на предвыборку и происходит обновление текущего смещения в файле. Как только очередь упреждающих вызовов заполняется, она перемещается в очередь модуля упреждающего кэширования на исполнение.

Рис. 4. Архитектура исполняющей системы

Модель исполнения приложения: в момент, когда вычислительный поток производит обращение к некоторым данным, эти данные уже находятся в файловом буфере, так как к ним уже было осуществлено обращение при исполнении упреждающего потока. Таким образом уменьшается количество блокирующих операций ввода/вывода и тем самым увеличивается эффективность выполнения приложения. Подобная схема эффективно работает для приложений с периодическими постоянными вызовами доступа к данным. Перед тем как исполнить любой запрос из очереди модуля упреждающего кэширования, происходит сравнение идентификаторов вызовов упреждающего и вычислительного потока. Если идентификатор упреждающего потока меньше, чем идентификатор вычислительного, то модуль просто удаляет этот вызов из очереди и переходит к выполнению следующего.

Пусть K – текущий размер очереди модуля упреждающего кэширования, T – среднее время исполнения одного запроса из очереди, а среднее время вычислений между двумя дисковыми вызовами пусть будет C. Предположим, что t – текущее время, тогда текущий идентификатор вызова вычислительного потока будет i, а упреждающего потока – j. Пусть A – временное предельное значение, а B – предельное ограничение дисковой очереди. Модуль упреждающего кэширования должен выполнить операцию упреждения, только если выполняются условия:

$$\left(C \times \left(j - i\right)\right) - \left(\left(K + 1\right) \times T\right) \le A, \quad (1)$$

$$\left(j - i\right) \le B. \quad (2)$$

Первое неравенство определяет, что подкачиваемый дисковый блок еще будет использоваться. Второе равенство определяет, что в буферной кэш-памяти есть еще место под следующие блоки. Обе эти предельные величины определяются текущей аппаратурой и пользователем и должны быть заданы заранее.

Заключение

Представленный вариант организации упреждающего кэширования на основе разделения потоков позволяет решить задачу определения последовательности будущих обращений к файлам для большого множества различных приложений с интенсивным обменом данными.

Список литературы

1. Ian Foster, Carl Kesselman. The Grid2 BluePrint for a new Computing Infrastructure. Second Edition. – Elsevier, 2003.

2. Curewitz K. Practical Prefetching via Data Compressing. In ACM Conference on Management of Data. – May 1993.

3. R. Hugo Patterson, Garth A. Gibson, M. Satyanarayanan. A Status Report on Research in Transparent Informed Prefetching. ACM Operating Systems Review, 1993.

4. Todd C. Mowry, Angela K. Demke, Orran Krieger. Automatic Compiler-Inserted I/O Prefetching for Out-of-Core Applications. Proceedings of the 1996 Symposium on Operating Systems Design and Implementation.

5. Fay Chang, Garth A. Gibson. Automatic I/O Hint Generation through Speculative Execution. Operating Systems Design and Implementation 2001.

Сахаров Илья Евгеньевич,
Академия ФСБ России,
научный сотрудник,
тел. 8-917-547-18-38, e-mail: siebox@mail.ru.

Ilya Sakharov,
research officer of Federal Security Service Academy,
telephone 8-917-547-18-38, e-mail: siebox@mail.ru.