Высокоуровневые средства программирования для процессора Cell

Ю.П. Сердюк

В статье дается обзор инструментальных средств, которые позволяют сделать программирование для процессора Cell более высокоуровневым. Рассматриваются системы MARS (Multicore Application Runtime System), Cacao VM for Cell, CellDotNet. Описываются принципы адаптации системы программирования MC# к вычислительным архитектурам (включая кластерные) на базе процессора Cell.

1. Введение

Процессор Cell Broadband Engine [1] представляет собой асимметричный многоядерный процессор, который объединяет процессор общего назначения IBM PowerPC (PPE) и восемь синергетических (synergistic) процессорных элементов (SPE), которые обеспечивают высокую пиковую производительность всей системы.

С другой стороны, состав средств программирования, включенных в IBM Cell SDK, ограничен компиляторами с языков C/C++ и Fortran, а также низкоуровневой библиотекой libspe2, содержащей функции для управления и взаимодействия с SPE. В состав этой библиотеки входят такие базовые функции как 1) *spe_context_create* – функция создания контекста (среды) для исполнения SPE-программ, 2) *spe_program_load* – функция загрузки программного кода в SPE, 3) *spe_context_run* – (синхронная) функция запуска SPE-программы на выполнение, 4) *spe_mfcio_get* и *spe_mfcio_put* – функции DMA-передачи данных, 5) *spe_in_mbox_write* и *spe_out_mbox_read* – функции записи и чтения сообщений в “почтовые ящики” (mail boxes), и др.

Однако, для массового внедрения и эффективного применения многоядерных архитектур, в том числе, на базе процессора Cell, необходимы новые высокоуровневые средства и языки параллельного программирования, которые могли бы быть использованы для регулярного программирования миллионами разработчиков. Основное требование к таким языкам состоит в том, что они должны быть достаточно прозрачны и формально “чисты”, чтобы для них могли быть разработаны соответствующие интегрированные инструменты для редактирования, компиляции, отладки, анализа производительности и т.д.

В данной работе дается обзор современных инструментальных средств, которые позволяют сделать программирование для процессора Cell более высокоуровневым, а значит, более продуктивным. Рассматриваются

1) система MARS (Multicore Application Runtime System) [2] – новый, более высокоуровневый интерфейс к базовой библиотеке libspe2 для программ на языке C;
2) Cacao VM for Cell [3] - виртуальная машина для языка Java на процессоре Cell;
3) система CellDotNet [4] – система программирования на языке C# для процессора Cell;
4) расширение системы программирования MC# [5] для процессора Cell – система на базе высокоуровневого языка программирования MC# для разработки приложений для Cell-архитектур, включая кластерные архитектуры.

2. Библиотека MARS (Multicore Application Runtime System)

Библиотека MARS предназначена для использования на многоядерных архитектурах, в которых имеется один главный (хост-) процессор, который управляет исполнением программ (процессов) на одном или более подчиненных блоках микрообработки (microprocessing units – MPUs).

Тем не менее, в MARS реализована концепция не централизованной относительно хост-процессора модели программирования, а модель, ориентированная на самоуправляемость бло-

ков микрообработки. Это достигается тем, что в каждый блок MPU в начале работы системы загружается ядро MARS, которое контролирует исполнение программ на данном MPU.

Базовым понятием при программировании с использованием библиотеки MARS, аналогично таким системам как Intel Threading Building Blocks и Microsoft Task Parallel Library, является понятие “задачи” (task). Ответственным за разбиение всей программы на части, включая “задачи”, которые будут исполняться на множестве MPU, является программист. Основное назначение библиотеки MARS состоит в том, что она позволяет создавать программисту произвольное количество задач для параллельного их исполнения на ограниченном количестве физических MPU.

Основные шаги по запуску задачи на MPU отражены в следующем фрагменте host-программы:

```
mars_context_create  ( &mars_context, 0, 0 );
mars_task_create     ( mars_context, &task_id, “Task”,
                       task_program_elf_image, 0      );
mars_task_schedule   ( &task_id, NULL, 0 );
mars_task_wait       ( &task_id, &task_exit_code );
mars_task_destroy    ( &task_id );
mars_context_destroy ( mars_context );
```

Рис. 1. Пример хост-программы с использованием библиотеки MARS

Передача данных между host-программой и MPU-программами, а также между отдельными MPU-программами осуществляется посредством DMA-функций, оформленных в виде функций *get* и *put* API библиотеки MARS.

В качестве средств синхронизации, библиотека MARS предоставляет обычные низкоуровневые конструкции типа мьютексов, семафоров и барьеров. Общее взаимодействие между host-программой и множеством задач, исполняющихся на MPU, возможно также посредством сигналов и событий. Для этого в библиотеке MARS есть ряд функций типа *mars_task_signal_send* и *mars_task_wait*, являющихся промежуточным слоем между пользовательской программой и функциями базовой библиотеки libspe2, работающими с “почтовыми ящиками” (mboxes).

Таким образом, библиотеку MARS можно охарактеризовать как более абстрактное API к базовой библиотеке libspe2, однако, остающееся довольно низкоуровневым средством программирования процессора Cell.

3. Виртуальная машина Cacao VM языка Java для процессора Cell

Работа по созданию виртуальной машины Java для процессора Cell, основанной на Cacao VM (http://www.cacaovm.org/), финансировалась фирмой IBM, закончилась созданием прототипа, права на программный код которого принадлежат данной фирме, и который, соответственно, не является открытым.

Основные особенности этой разработки состоят в следующем:

1) Оригинальная виртуальная машина Cacao VM модифицирована таким образом, что ее JIT-компилятор способен порождать машинный код для SPE-элементов в процессе выполнения основной программы на PPE. За основу был принят PowerPC JIT-компилятор, поскольку множества машинных инструкций основного процессора PowerPC и SPE во многом сходны. Код для SPE, который генерирует JIT-компилятор, удовлетворяет следующим требованиям: a) доступ к локальной памяти (LS) SPE-элементов выровнен на 16-байтовую границу, b) для достижения максимальной эффективности исполнения, код имеет наименьшее количество ветвлений, c) код использует уникальные конструкции из множества инструкций SPE, обеспечивающие его высокую производительность. Кроме непосредст-

венно порождения кода, JIT-компилятор обеспечивает распределение регистров для оттранслированных SPE-программ.

2) В Cacao VM для Cell реализован одновременный доступ из PPE- и SPE-потоков к объектам, находящемся в “куче” (heap) – базовом хранилище объектов исполняющейся Java-программы. Эффективность этого доступа обеспечивается использованием DMA-функций. Сборка мусора среди объектов, созданных в SPE-программе, в данном прототипе не реализована.

3) В виду ограниченности размера LS, в локальной памяти SPE-элементов организовано хранение только тех откомпилированных методов, которые используются в текущий момент или которые скоро будут использованы. Остальные методы для SPE кэшируются в главной памяти, для перемещения которых в LS и обратно разработан специальный механизм на базе SPU Software Managed Cache library. Методы, которые предназначены для исполнения на SPE, помечаются программистом в Java-программе специальными тегами. Эта идея использована также при адаптации системы программирования MC# к процессору Cell (см. об этом ниже).

4) В Cacao VM для Cell поддержан вызов из SPE-программ функций, котрые должны быть выполнены на PPE-элементе или других SPE-элементах. В частности, SPE-методы могут печатать значения на консоли, вызывая методы ввода-вывода на PPE.

4. Система программирования CellDotNet

Система программирования CellDotNet позволяет исполнять программы, написанные на языке C#, на процессоре Cell. Хотя, как и в случае с языком Java, исполнение таких программ не настолько эффективно, как исполнение программ, написанных на языке C, однако, существует достаточное число приложений, написанных на байт-кодовых языках, например, финансовых приложений, которые будучи портированными на архитектуру Cell, будут более быстродействующими в несколько раз.

Система CellDotNet работает поверх системы Mono – свободной реализации платформы .NET для Unix-подобных систем, и состоит из JIT-компилятора и подсистемы поддержки исполнения программ (runtime-системы), полностью написанных на языке C#.

Основные шаги необходимые для запуска метода на SPE-элементе показаны в следующем фрагменте программы:

```
private delegate void computeDelegate ( < parameters > ){

public void run ( )   {

 computeDelegate   fun = compute;
 CompileContext cc = new CompileContext ( fun.Method );
 cc.PerformProcessing ( CompileContextState.S8Complete );
 int result = (int) SpeContext.RunProgram ( cc, < args > );

}

public static int compute ( < parameters > ) { … }
```

Рис. 2. Пример запуска SPE-программы в CellDotNet

Этот фрагмент демонстрирует а) этап JIT-компиляции метода *compute*, предназначенного для исполнения на SPE-элементе, с помощью метода *PerformProcessing* класса *CompileCon-*

text, и b) этап запуска откомпилированного метода на исполнение с помощью метода *RunProgram* класса *SpeContext*. Метод *RunProgram* является синхронным, т.е., он заканчивается только после того, как завершается исполнение соответствующего метода на SPE-элементе. Поэтому для одновременного использования нескольких SPE, соответствующие шаги для каждого SPE исполняются в отдельно запускаемом потоке. После завершения работы SPE-метода, функция *RunProgram* возвращает вычисленный результат, а также диагностическую информацию о корректности завершения вызова.

Для передачи в SPE-методы массивов данных, а также их возврата оттуда, используется класс *AlignedMemory* из библиотеки CellDotNet. Его необходимость вызвана требованием при применении DMA-функций использовать только области памяти, выровненные по 16-байтной границе. Внутри SPE-методов, данные из LS в общую память и обратно передаются посредством функций *Get* и *Put* класса *Mfc*.

Текущая реализация CellDotNet имеет существенные ограничения. Например, в ней не поддержаны некоторые базовые структуры данных, в частности, тип *double*, а также нет средств для прямого взаимодействия между собой методов, исполняемых на различных SPE-элементах.

5. Адаптация системы программирования MC# к процессорным архитектурам Cell

Система программирования MC# основана на высокоуровневом, объектно-ориентированном языке MC#, который является расширением языка C# и предназначен для разработки параллельных (исполняемых на многоядерных процессорах) и распределенных (исполняемых на кластере) приложений. Эта система базируется на платформе .NET, а потому имеет реализации как для ОС Windows, так и для ОС Linux (под системой Mono).

Базой для адаптации системы программирования MC# к вычислительным архитектурам на базе процессора Cell является система CellDotNet, которая представляет собой высокоуровневый интерфейс к базовой библиотеке *libspe2* пакета IBM CellSDK, позволяющий управлять и взаимодействовать с SPE-процессами.

Основным механизмом параллельности в языке MC# являются

a) *асинхронные* (*async*) методы, вызов которых приводит к порождению нового локального потока, в рамках которого выполняется тело данного метода, и

b) *перемещаемые* (*movable*) методы, вызов которых аналогичен вызову *async*-методов, за исключением того, что новый поток может быть порожден на удаленной машине, например, на наименее загруженном узле кластера.

Средствами взаимодействия между параллельными процессами (*async*- и *movable*-методами) в языке MC# являются *каналы* и *обработчики*: по каналам может быть послан произвольный набор значений/объектов, который может быть принят (возможно, с некоторой предобработкой) соответствующим обработчиком.

В библиотеке CellDotNet, в классе *SpeContext* имеется несколько методов *RunProgram*, с помощью которых может быть осуществлен запуск программ на SPE. Схематично, типичный вызов метода *RunProgram* выглядит так:

```
<type> result = (<type>) SpeContext.RunProgram ( function, < args > );
```

Рис. 3. Вызов метода *RunProgram* в CellDotNet

где *function* есть делегат, значением которого является вызываемая функция. Особенность данного вызова состоит в том, что он является синхронным – вызов *RunProgram* заканчивается только тогда, когда завершается исполнение соответствующей функции на SPE-элементе.

Базовая идея адаптации системы программирования MC# к вычислительным архитектурам на базе процессора Cell состоит во введении дополнительного модификатора *spe*, который может быть использован в определении метода, предназначенного для исполнения на SPE.

Ниже приведен полный текст программы на языке MC#, предназначенной для параллельного вычисления на SPE-элементах приближенного значения числа π методом Монте Карло (вспомогательный класс *RandomSingle*, служащий для генерации случайных чисел с плавающей точкой одинарной точности, опущен).

```
using System;
public  class   MonteCarloPi    {
  public static int original_seed  = 113;
  public static void Main ( String[] args )      {
   int    i;
   int    seed;
   int    iterations   = Convert.ToInt32  ( args [ 0 ] );
   int    P  = Convert.ToInt32( args [1] ); // Number of (SPE-) processors
   MonteCarloPi   mcp   = new MonteCarloPi() ;
   for ( i = 0 ; i < P ; i++ )   {
    if ( i % 2 == 0 )                 //   Prepare the different seeds
     seed = original_seed -  i / 2;   //  for each of asyncspe-methods
    else
     seed = original_seed  + ( i + 1 ) / 2;
    mcp.aintegrate(seed, iterations/P, mcp.sendResult);// Run the spe-
                                                       // methods
   }
   float   pi = 0.0F;
   for ( i = 0; i < P; i++ )                   //   Get the results
    pi += (float) getResult ? ();
   Console.WriteLine ( "pi = " + ( pi / 4.0 ) );
  }
  public async aintegrate (int seed, int iterations, channel (float)
                           sendResult                                  ) {
   sendResult ! ( integrate ( seed, iterations ) );
  }
  public spe float integrate (int seed, int iterations ) {
   RandomSingle R = new RandomSingle ( seed );
   int under_curve = 0;
   for ( int count = 0; count < iterations; count++ )  {
     float x = R.nextFloat();
     float y = R.nextFloat();
     if (x * x + y * y <= 1.0f)
      under_curve++;
   }
   return  ( ( (float)under_curve / Num_samples) * 4.0F );
  }
  public handler getResult  float () & channel sendResult  (float r)   {
   return ( r );
  }
}
```

Рис. 4. Программа вычисления числа π методом Монте Карло на языке MC#

Транслятор языка MC#, встречая в тексте программ модификатор *spe* для некоторого метода, производит генерацию всех необходимых описаний, необходимых для запуска данного метода на SPE. В частности, при передаче в качестве аргументов вызова массива, будут сгенерированы соответствующие вызовы DMA-функций для передачи массивов из общей памяти в локальную.

Соответственно, чтобы получить распределенную MC#-программу для исполнения на кластере с процессорами Cell, достаточно в тело *movable*-методов вставить вызовы *spe*-методов.

Таким образом, от прикладного программиста оказываются скрытыми подробности использования библиотеки CellDotNet, и он остается в рамках единой асинхронной модели программирования, принятой в языке MC#.

В рамках проекта адаптации системы программирования MC# к процессорным архитектурам Cell, предполагается также поддержать конструкцию *Future*, используемую в библиотеке Parallel FX фирмы Microsoft (http://blogs.msdn.com/pfxteam) и языке X10 (http://x10-lang.org/). В частности, предполагается введение конструкции *FutureSPE*, с помощью которой запуск вычислительного процесса на SPE-элементе и получение результирующего значения может быть оформлено следующим образом (на примере метода *integrate* класса *MonteCarloPi*):

```
FutureSPE<float> pi = new FutureSPE<float>{MonteCarloPi.integrate
(113, 10000);};

float   result = (float) pi.force();
```

Рис. 5. Пример использования конструкции *FutureSPE* в программах на MC#

В ходе экспериментов с системами MC# и CellDotNet, нами было реализовано несколько приложений, включая комбинаторное приложение расстановки ферзей на доске размером N x N, а также вычисление множества Мандельброта (фрактала).Тестирование проводилось на сервере QS22, имеющем 2 процессора Cell, т.е., 4 логических основных процессора (PPE) и 16 SPE-элементов. В качестве системного ПО, использовался дистрибутив Fedora Core 9 системы Linux и система Mono версии 1.9.1 из этого дистрибутива. Также были сделаны некоторые доработки самой системы CellDotNet для обеспечения ее работоспособности на серверах серии QS (в оригинале система CellDotNet разрабатывалась и тестировалась только на Play Station 3). Ниже представлен график зависимости времени выполнения от количества использованных SPE-процессоров для приложения NQueens (N = 17).

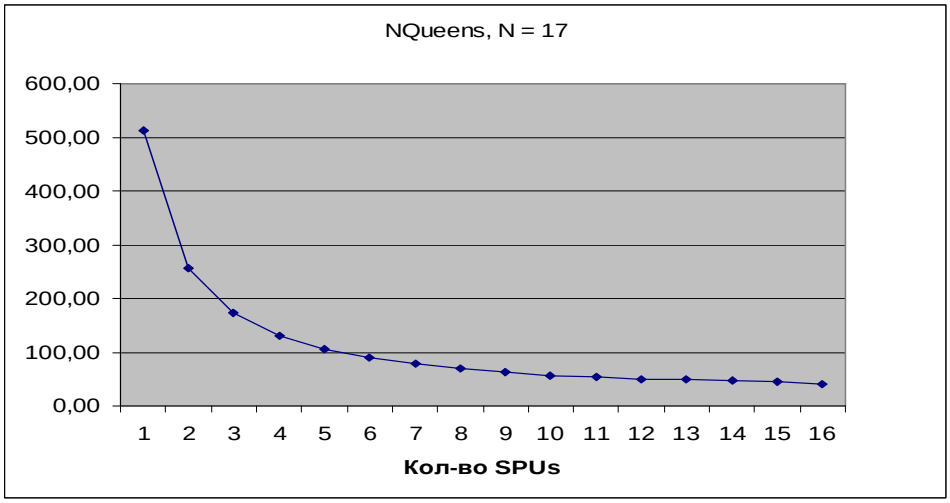

Рис. 6. График зависимости времени выполнения от количества SPE-процессоров для программы NQueens

6. Заключение.

Системы программирования на базе языков Java и C# способны повысить интерес к использованию платформы Cell в различных областях практических приложений. В частности,

система программирования MC# значительно облегчает создание параллельных приложений, использующих одновременно PPE- и SPE-процессоры.

Основными направлениями по дальнейшему развитию библиотеки CellDotNet и системы программирования MC# для Cell являются:

1) расширение перечня конструкций и списка типов данных, поддерживаемых библиотекой CellDotNet;
2) исследование и реализация механизмов доступа к массивам в основной памяти из SPE-программ;
3) исследование и реализация механизмов взаимодействия SPE-методов посредством каналов и обработчиков.

Литература

1. Kahle J., Day M., Hofstee H., Johns C., Maurer T., Shippy D. Introduction to the Cell Multiprocessor // IBM Journal of Research & Development. – 2005. –Vol. 49, N. 4/5. –P. 589–604.
2. MARS: ftp://ftp.uk.linux.org/pub/linux/Sony-PS3/mars/
3. Cacao VM for Cell: http://vergiss-blackjack.de/diploma-thesis_georg-sorst_java-on-cell.pdf.
4. CellDotNet: http://code.google.com/p/celldotnet/.
5. MC#: http://www.mcsharp.net.