Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное  
учреждение высшего образования   
«Ивановский государственный энергетический университет  
имени В.И. Ленина»

Факультет информатики и вычислительной техники

Кафедра Высокопроизводительных вычислительных систем

**Отчет по курсовой работе**

**по технологиям параллельного программирования**

Системы параллельного программирования

Выполнил:

Прохоров А.П.

Проверила:

Чернышева Л.П.

**Технологии параллельного программирования**

**1. PVM**

**2. MPC**

**3. DVM**

**4. LINDA**

**1. PVM**

PVM (параллельная виртуальная машина) - это побочный продукт продвижения гетерогенных сетевых исследовательских проектов, распространяемый авторами и институтами, в которых они работают.

Общими целями этого проекта являются исследование проблематики и разработка решений в области гетерогенных параллельных вычислений. PVM представляет собой набор программных средств и библиотек, которые эмулируют общецелевые, гибкие гетерогенные вычислительные структуры для параллелизма во взаимосвязанных компьютерах с различными архитектурами.

Главной же целью системы PVM является обеспечение возможности совместного использования группы компьютеров совместно для взаимосвязанных или параллельных вычислений.

Система PVM состоит из двух частей. Первая часть - это ``демон'' под названием pvmd3 - часто сокращается как pvmd, - который помещается на все компьютеры, создающие виртуальную машину. (Примером программы-демона может быть почтовая программа, которая выполняется в фоновом режиме и обрабатывает всю входящую и исходящую электронную почту компьютера). Разработан pvmd3 таким образом, чтобы любой пользователь с достоверным логином мог инсталлировать его на машину. Когда пользователь желает запустить приложение PVM, он прежде всего создает виртуальную машину. После этого приложение PVM может быть запущено с любого UNIX-терминала на любом из хостов. Несколько пользователей могут конфигурировать перекрывающиеся виртуальные машины, каждый пользователь может последовательно запустить несколько приложений PVM.

Вторая часть системы - это библиотека подпрограмм интерфейса PVM. Она содержит функционально полный набор примитивов, которые необходимы для взаимодействия между задачами приложения. Эта библиотека содержит вызываемые пользователем подпрограммы для обмена сообщениями, порождения процессов, координирования задач и модификации виртуальной машины.

Ниже приведена программа PVM hello - простой пример, который иллюстрирует базовую концепцию программирования PVM. Эта программа рассматривается как запускаемая вручную; после вывода на экран своего идентификатора задачи (полученного с помощью pvm\_mytid()) она порождает копию другой программы под названием hello\_other, используя функцию pvm\_spawn(). Успешное порождение заставляет программу выполнить блокирующий прием с помощью pvm\_recv(). После приема сообщения программа выводит на экран сообщение, посланное ей абонентом - так же как и свой идентификатор задачи; содержимое буфера извлекается из сообщения применением pvm\_upsksrt(). Заключительный вызов pvm\_exit ``выводит'' программу из системы PVM:

**2. MPC**

Язык параллельного программирования mpC был разработан в конце 90х в Институте системного программирования РАН. В языке mpC, который является расширением языка С, вводится понятие вычислительного пространства, которое определяется как некоторое доступное для управления множество виртуальных процессоров. Основным понятием mpC является понятие сетевого объекта или просто сети. Сеть является областью вычислительного пространства, которая может быть использована для вычисления выражений и выполнения операторов.

Программист на mpC не может управлять тем, сколько процессов составляют программу и на каких компьютерах эти процессы выполняются. Это делается внешними по отношению к языку средствами. Исходный код на mpC управляет лишь тем, какие именно вычисления выполняются каждым из процессов, составляющих программу.

Основное внимание в языке *mpC* уделяется средствам, позволяющим максимально облегчить разработку как можно более эффективных программ для решения задач на обыкновенных сетях компьютеров.

Библиотечная узловая функция MPC\_Printf языка mpC гарантирует вывод приветствия на терминал пользователя от каждого компьютера, участвующего в процессе параллельной программы.

*#include <mpc.h>*

*int [\*]main()*

*{*

*MPC\_Printf(" Hello, world! \n");*

*}*

**3. DVM**

DVM-система предназначена для создания переносимых и эффективных вычислительных приложений на языках C-DVM и Fortran-DVM для параллельных компьютеров с различной архитектурой.

Аббревиатура DVM соответствует двум понятиям: Distributed Virtual Memory и Distributed Virtual Machine. Первое отражает наличие единого адресного пространства. Второе отражает использование виртуальных машин для двухступенчатой схемы отображения данных и вычислений на реальную параллельную машину.

В систему DVM также входят библиотека поддержки LIB-DVM, DVM-отладчик, предсказатель выполнения DVM-программ, анализатор производительности DVM-программ.

Система разработана в [Институте прикладной математики им. М.В.Келдыша РАН](https://parallel.ru/russia/organizations.html#ipm).

В DVM-системе программисту предоставляются следующие возможности спецификации параллельного выполнения программы:

* распределение элементов массива между процессорами;
* определение циклов, витки которых могут выполняться параллельно (витки циклов распределяются между процессорами в соответствии с распределением данных);
* спецификация параллельно выполняющихся секций программы (параллельных задач) и отображение их на процессоры;
* организация эффективного доступа к удаленным (расположенным на других процессорах) данным;
* организация эффективного выполнения редукционных операций — глобальных операций с расположенными на различных процессорах данными (таких, как их суммирование или нахождение их максимального или минимального значений);
* спецификация регионов – специальных конструкций языка, состоящих из последовательных

**Состав DVM-системы**

DVM-система состоит из следующих компонент:

* Компилятор Fortran-DVMH
* Компилятор C-DVMH
* Библиотека LIB-DVMH
* Средства функциональной отладки DVMH-программ
* Средства отладки эффективности DVMH-программ

Эти компоненты выполняют следующие функции:

* Компилятор Fortran-DVMH превращает параллельную программу в программу на языке Фортран, использующую стандартные технологии программирования OpenMP и CUDA и расширенную вызовами функций Lib-DVMH.
* Компилятор C-DVMH превращает параллельную программу в программу на языке Си, использующую стандартные технологии программирования OpenMP и CUDA и расширенную вызовами функций Lib-DVMH.
* Библиотека Lib-DVMH — это система поддержки выполнения DVMH-программ (написанных на языках Fortran-DVMH или C-DVMH). Функции Lib-DVMH используют стандартную коммуникационную систему MPI и технологию CUDA.
* Средства функциональной отладки DVMH-программ обеспечивают выполнение программы на рабочей станции в специальном режиме проверки DVMH-указаний, а также выполнение программы на параллельном компьютере в специальном режиме, когда промежуточные результаты выполнения сравниваются с результатами эталонной трассировки (например, результатами последовательного выполнения). Кроме того, для отладки программ, использующих в качестве ускорителей графические процессоры, имеется режим сравнения результатов выполнения региона на ЦПУ и ГПУ.
* Анализатор производительности определяет характеристики эффективности выполнения параллельной программы. Он с заданной степенью детализации предоставляет пользователю информацию о производительности всей DVMH-программы и различных ее фрагментов.

#### Использование DVM на кластере

Необходимо создать поддиректорию, в которой будет находится DVM-программа, и сделать ее текущей. Затем по команде "**dvminit**" надо получить номера установленных версий DVM-системы (сейчас это 315 и 315n) и инициализировать одну из них командой "**dvminit номер\_версии**" (рекомендуется использовать последнюю).

#### Трансляция и запуск программ

Программы на языках C-DVM и Fortran-DVM имеют расширения .cdv и .fdv, соответственно. Оттранслировать DVM-программу можно с помощью команды "**dvm c program.cdv**" или "**dvm f program.fdv**". Если нет ошибок, генерируется исполняемая программа (program), работающая в параллельном режиме поверх MPI.

Запустить программу на выполнение можно командой "**dvm run M N program**". Здесь MxN - размерность решетки процессоров (программа запускается на M\*N процессорах).

**4. LINDA**

Идея построения системы Linda исключительно проста, а потому красива и очень привлекательна. Параллельная программа есть множество параллельных процессов, и каждый процесс работает согласно обычной последовательной программе. Все процессы имеют доступ к общей памяти, единицей хранения в которой является кортеж. Отсюда происходит и специальное название для общей памяти - пространство кортежей. Каждый кортеж это упорядоченная последовательность значений. Например,

( "Hello", 42, 3.14 ),

( "P", 5, FALSE, 97, 1024, 2),

( "worker", 5 ) .

Первый элемент кортежа всегда является символьной строкой и выступает в роли имени кортежа. Так первый кортеж предыдущего примера состоит из имени ("Hello"), элемента целого типа (42) и вещественного числа (3.14). Во втором кортеже кроме имени "P" есть элемент целого типа (5), элемент логического типа (FALSE) и три целых числа. Последний кортеж состоит из двух элементов: имени ("worker") и целого числа (5). Количество элементов в кортеже может быть любым.

**Функции Linda**

1. Функция OUT помещает кортеж в пространство кортежей.
2. Функция IN ищет подходящий кортеж в пространстве кортежей, присваивает значения его элементов элементам своего параметра-кортежа и удаляет найденный кортеж из пространства кортежей.
3. Функция READ отличается от функции in лишь тем, что выбранный кортеж не удаляется из пространства кортежей. Все остальное точно так же, как и у функции in. Этой функцией удобно пользоваться в том случае, когда значения переменных менять не нужно, но к ним необходим параллельный доступ из нескольких процессов.
4. Функция EVAL похожа на функцию out. Разница заключается лишь в том, что дополнительным элементом кортежа у eval является функция пользователя. Для вычисления значения этой функции система Linda порождает параллельный процесс, на основе работы которого она формирует кортеж и помещает его в пространство кортежей.

**Синхронизация потоков**

Не имея в системе Linda никаких явных средств для синхронизации процессов, совсем не сложно их смоделировать самому. Предположим, что в некоторой точке нужно выполнить барьерную синхронизацию N процессов. Какой-то один процесс, например, стартовый, заранее помещает в пространство кортеж ("ForBarrier", N). Подходя к точке синхронизации, каждый процесс выполняет следующий фрагмент, который и будет выполнять функции барьера:

in( "ForBarrier", formal Bar);

Bar = Bar - 1;

if( Bar != 0 ) {

out( "ForBarrier", Bar);

read( "Barrier" );

} else

out( "Barrier" );

Если кортеж с именем "ForBarrier" есть в пространстве, то процесс его изымает, в противном случае блокируется до его появления. Анализируя второй элемент данного кортежа, процесс выполняет одно из двух действий. Если есть процессы, которые еще не дошли до данной точки, то он возвращает кортеж в пространство с уменьшенным на единицу вторым элементом и встает на ожидание кортежа "Barrier". В противном случае он сам помещает кортеж "Barrier" в пространство, который для всех является сигналом к продолжению работы.