Лабораторная работа №1

Получение первичных навыков организации распределённого выполнения программ с использованием библиотеки PVM.

1. Цель работы: изучение средств, представляемых PVM для организации вычислительных кластеров, функций библиотеки PVM, предназначенных для организации распределённых вычислений на ПК вычислительного кластера.

2. Теоретическое введение.

2.1. Технология создания PVM-программ

Технология создания PVM-программ предполагает реализацию последовательных программ на хостах вычислительного кластера. Каждая последовательная программа, реализуемая на одном из хостов кластера, содержит обращения к подпрограммам библиотеки PVM. Эти обращения позволяют реализовать управляемый запуск задач на этапах кластера некоторой главной (родительской) задачей и обмен сообщениями между выполняемыми одновременно задачами, что обеспечивает возможность получения требуемого решения. Каждая из последовательных программ транслируется на тех хостах, которые будут включены в состав вычислительного кластера (параллельной виртуальной машины – PVM), т. е. Реализуется раздельная трансляция программ. Исполняемые файлы программ (части одной общей параллельной программы) размещаются в определённых каталогах на своих хостах.

Для инициализации выполнения программ на хостах кластера запускается главная задача, которая производит запуск остальных подчинённых задач.

При запуске подчинённых задач система (PVM) ищет исполняемые файлы в каталоге /PVM3/bin/PVM.arch, куда разработчиком и должны помещаться откомпилированные программы.

Взаимодействие между выполняемыми параллельно (на разных хостах кластера) процессами осуществляется с использованием сообщений.

2.2. Определение (задание) конфигурации вычислительного кластера (параллельной виртуальной машины).

Прежде, чем определить процесс конфигурирования виртуальной машины (в дальнейшем, ВМ), необходимо определить две компоненты PVM:

1) Демон PVM – процесс управления выполнением PVM-программ. Эти процессы запускаются на всех хостах, входящих в состав ВМ. Если разработчик собирается инициализировать выполнение PVM-программ на нескольких хостах, он должен запустить демон на тех хостах, на которых будет выполняться параллельное приложение. Т. о. запуск демонов (управляющих процессов) на тех хостах, где будет выполняться программа и является создание ВМ.

2) Набор подпрограмм, входящих в библиотеку PVM, реализующих определённые операции, обеспечивающие функционирование хостов в вычислительном кластере и выполнение параллельных программ.

Кроме введённых понятий компонент PVM необходимо определять понятие идентификитора задачи (Task Identifier – TId). Каждому процессу, входящему в состав PVM-программы, локальным демоном назначается свой уникальный идентификатор задачи, который используется для адресации сообщений, передаваемых между хостами в ВМ.

Задание конфигурации вычислительного кластера осуществляется с консоли PVM, которая запускается на одном из хостов ВМ командой #PVM.

PVM-консоль – это программа, которая позволяет работать с параллельной виртуальной машиной в интерактивном режиме. После выполнения команды запуска консоли конфигурирование состава ВМ осуществляется в её командной строке. При включении в конфигурацию ВМ нового хоста, демон PVMd загружается на нём и хост может выполнять

находящиеся на нём программы, запускаемые главной задачей. Конфигурирование состава ВМ может осуществляться с одного из хостов, входящего в её состав, путём выполнения соответствующих команд, подключающих или отключающих хосты от ВМ. Приведённые ниже команды позволяют формировать вычислительный кластер вручную (с консоли) и контролировать его состав:

1) add <хост> successful – добавляет хост с указанным именем в состав ВМ;
2) delete <хост> – удаляет хост с указанным именем из состава ВМ;
3) id – выводит идентификатор задачи, реализуемый на данном хосте;
4) conf – выводит список хостов, входящих в ВМ;
5) quit – завершает работу с PVM-консолью без удаления ВМ и без завершения работы параллельных программ.

Следует отметить, что при выполнении команды conf PVM выводит наряду с именами хостов, входящих в состав ВМ, и идентификатор демона ВМ (DTI d), управляющего выполнением программ на данном хосте (не следует путать с идентификатором задачи Tid, который может быть определён командой ID в режиме консоли на соответствующем хосте).

После конфигурирования состава ВМ на одном из хостов м/б запрещена главная (родительская) задача, которая будет инициализировать выполнение программ на других хостах ВМ.

2.3. Организация распределённого выполнения параллельных программ на ПК вычислительного кластера с обменом сообщениями между ними.

Организация распределённого выполнения параллельных программ предлагает процессами (в частности, их инициализацию) на хостах, входящих в состав кластера. Управление процессами (их инициализация и завершение) осуществляется путём вызова соответствующих программ, входящих в состав библиотеки PVM. Общий формат вызовов этих подпрограмм, а также назначение параметров подпрограмм, приведённых ниже:

1) подпрограмма pvm_mrtid() позволяет определить идентификатор задачи (TID), выполняемый на данном хосте. Вызов этой подпрограммы является обязательным на каждом этапе не только потому, что она позволяет определить TID, но и потому, что она подключает выполняемую на хосте программу к PVM;

2) подпрограмма pvm_exit() сообщает локальному демону PVMd о том, что процесс завершает работу с PVM (демон освобождает ресурсы, выделенные этому домену). Вызов подпрограммы pvm_exit() возвращает код завершения программы (отрицательное значение соответствует ошибке);

3) подпрограмма pvm_spawn() запускает ntask копий исполняемого файла, имя которого содержится в первом параметре вызова этой подпрограммы. Общий формат вызова подпрограммы следующий: int numt = pvm_spawn (n_task, arg, flag, where, ntask, tids).

Параметры task, arg, flag, where, ntask являются входными, tids – выходной. Первый параметр (mun char) задаёт имя запускаемой программы на соответствующем хосте кластера. Второй параметр указывает на символьный массив аргументов, передаваемый в запускаемую программу. Если аргументов у задачи нет, null должен быть вторым элементом массива. Параметр flag (тип целый) используется для задания режимов выполнения запускаемого процесса. Для простоты этот параметр будет предполагаться равным нулю.

Параметр where (mun char) определяет символьное имя хоста, на котором должна запускаться программа с указанным в первом параметре именем. Параметр ntask имеет целый тип и определяет количество запускаемых копий одной программы на соответствующем хосте. Параметр tids является выходным и возвращает массив идентификаторов успешно запущенных задач (TID). В случае, если запуск задачи на соответствующем хосте не произошёл, переменная numt содержит в себе код ошибки выполнения (PVM Bad Param, PVM No Host, PVM No File, PVM No Mem, PVM Sys Err, PVM Out Of Res). После выполнения подпрограммы PVM_spawn переменная numt содержит количество фактически запущенных процессов.

При выполнении задач на хостах вычислительного кластера между ними осуществляется обмен сообщениями. В PVM реализована следующая модель передачи сообщений:

1) задачи обмениваются сообщениями произвольной длины. Для обмена данными между хостами они (данные) преобразуются в формат XDR (внешнее представление данных);

2) сообщению в момент передачи присваивается числовой идентификатор, назначаемый разработчиком. Идентификаторы используются для различения сообщений, передаваемых между хостами;

3) источник сообщения не ожидает от адресата подтверждения приёма сообщения;

4) перед приёмом сообщение буферизируется на стороне адресата. Память для буферов выделяется динамически;

5) последовательность сообщений от одного источника к одному и тому же адресату сохраняется.

В PVM передача сообщений выполняется за три шага. На первом создаётся буфер передачи, после чего данные для сообщения упаковываются в буфер. При этом для упаковки разнотипных данных используются различные подпрограммы, входящие в библиотеку PVM. После размещения данных в буфере осуществляется отправка сообщения адресату. В случае, если сообщение используется только для синхронизации, но не для передачи данных, первые два шага м/б пропущены и осуществляется только передача синхронизирующих сообщений.

Создание буфера для передачи данных осуществляется подпрограммой PVM_init send, имеющей следующий формат:

Int bufid = PVM_initsend (int encoding).

При этом создаётся один активный буфер, идентификатор которого возвращается в параметре bufid. Параметр encoding определяет метод кодирования упаковываемых данных: 0 – метод, используемый по умолчанию; 1 – передача данных без кодирования, используемая в однородном (по архитектуре) кластере; 2 – данные не перемещаются в буфер, а остаются в памяти, откуда и передаются адресату.

Подпрограммы упаковки позволяют разместить данные непосредственно в активном буфере. Для кодирования каждого типа данных используется своя подпрограмма, имеющая три входных параметра. Упаковывающими подпрограммами являются:

PVM – pkdouble (double*cp, int nitem, int stride);

PVM – pkfloat (float*cp, int nitem, int stride);

PVM – pkint (int*cp, int nitem, int stride);

PVM – pklong (long*cp, int nitem, int stride);

PVM – pkshort (short*cp, int nitem, int stride).

Первый параметр в вызываемых подпрограммах указывает на элемент передаваемых данных, второй параметр определяет количество упаковываемых элементов и третий параметр – способ упаковки (1 – упаковка непрерывного массива данных, 2 – упаковка элементов массива через фич).

Соответствующие программы распаковки данных имеют параметры, аналогичные по назначению выше приведённым (для упаковки). Их имена в библиотеке PVM определены следующим образом:

PVM – upk double, PVM – upk float, PVM – upk int, PVM – upk long, PVM – upk short.

Распаковка данных выполняется после передачи синхронизирующего сообщения, которая осуществляется путём вызова подпрограммы PVM _send в следующем формате:

Int ierr = PVM_send (int tid, int msqtag); где tid – идентификатор адресата (TID адресата), msqtag – неотрицательное целое значение, определяющее номер передаваемого сообщения (т. к. модель передачи сообщений не предполагает ожидания подтверждения приёма, а их количество (порядок следования) необходимо контролировать).

Ожидание сообщения осуществляется вызовом подпрограммы PVM_recv в следующем формате:

Int id = PVM_recv (int tid, int msqtag), где tid определяет идентификатор задачи, которая является источником сообщения, msqtag характеризует номер передаваемого сообщения. Значение параметра tid, равное – 1, в функции сообщения.

Важной подпрограммой, обеспечивающей обмен сообщениями между задачами на различных хостах, является подпрограмма, которая позволяет подчинённой задаче определить tid

родительской задачи. Tid родительской задачи необходим для того (если синхронизация идёт с этой задачей), чтобы синхронизируемая задача могла в подпрограмме PVM-recv указать это значение, как идентификатор процесса, от которого она ждёт синхронизации. Подпрограмма, позволяющая определить tid родительской задачи имеет следующий формат:

Int tid = PVM-parent().

Пример программы, организующей распределённое выполнение программы нескольких хостах вычислительного кластера и обмен данными между ними через буфер, приведён ниже:

```
#include “PVM3. h”

main ( )
{
int ierr1, ierr2, buf, ierr3;
int tid1, tid2, msqtag;
int param1;

tid1 = PVM_mytid();
printf(“tid для главной задачи”, tid1);
ierr1 = PVM_spawn (“prog1”, (char**) 0, 0, “host1”,tid2); //промежуточные вычисления по подготовке данных, передаваемых в программу “prog1” на хосте “host1”
buf = PVM_init send (0); // создание активного буфера – 1 кс.
ierr2 = PVM_print (param1, 1, 1); msqtag = 1;
ierr3 = PVM_send (tid2, msqtag); //возможное ожидание подтверждения приёма синхрособщения.
PVM_exit ();
}
```

Текст вызываемой программы “prog1”, функционирующей на хосте”host1”

```
#include “PVM3. h”
main ()
{
int ierr
int tid1, tid2
int param1;
tid2 = PVM_mytid (); // определение своего tid
tid1 = PVM_parent (); // определение tid запустившей родительской задачи
msqtag = 1;
ierr1 = PVM_recv (tid1, msqtag);
PVM_upk ind (param1, 1, 1);
Printf (“Принятый параметр”, param1);
PVM_exit ();
}
```

3. Задание на работу.

Задание выполняется в соответствии с вариантом, назначенным преподавателем.

1 Вариант

Выполнить расчёт произведения матриц 3x3, заданных в следующем виде:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Вычисление выполняется на трёх хостах:

1) на первом (родительская задача) ищется произведение первой строки матрицы A на все столбцы матрицы B – первая строка результирующей матрицы;

2) на втором и третьем хостах соответственно ищется произведение второй и третьей строк матрицы A на столбцы матрицы B (эти параметры передаются на эти хосты родительской задачей, результат вычисления возвращается обратно в родительскую задачу).

2 Вариант

Вычислить определитель матрицы 3x3, заданный в следующем виде (методом треугольников):

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Вычисления выполняются на двух хостах:

1) на первом (родительская задача)осуществляется подсчёт суммы произведений элементов главной диагонали и элементов, расположенных параллельно ней;

2) на втором хосте осуществляется подсчёт сумм произведений элементов побочной диагонали и элементов, расположенных параллельно ней;

3) результирующая значение определителя вычисляется родительской задачей.

3 Вариант

Вычислить определитель матрицы 4x4, в соответствии со следующей формулой:

$$|A| = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{123} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = a_{11} \cdot \begin{bmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{bmatrix} + a_{11} \cdot \begin{bmatrix} a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{bmatrix} + a_{13} \cdot \begin{bmatrix} a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \\ a_{41} & a_{42} & a_{44} \end{bmatrix} +$$

$$+ a_{14} \cdot \begin{bmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}$$

Для вычисления определителя созда1тся вычислительный кластер, состоящий из двух хостов. На первом хосте, где выполняется главная задача, осуществляется ввод данных и

вычисление двух первых слагаемых результирующей суммы. На втором хосте осуществляется вычисление двух последующих слагаемых результирующей суммы. Вычисление конечного результата вычисляется на первом хосте.

4. Контрольные вопросы.

4.1. Охарактеризовать технологию создания PVM-программ.
4.2. Основные компоненты PVM.
4.3. Команды PVM-консоли для создания вычислительного кластера.
4.4. Подпрограммы организации распределённого выполнения программ на хостах вычислительного кластера.
4.5. Подпрограммы организации взаимодействия между задачами, выполняемыми на хостах вычислительного кластера, с помощью сообщений.
4.6. Модель обмена сообщениями PVM.