ФУНКЦИИ БИБЛИОТЕКИ MPE И ЛОГФАЙЛЫ.

Библиотека MPE (Multi-Processing Environment) содержит процедуры, которые находятся "под рукой" и облегчают написание, отладку и оценку эффективности MPI–программ. MPE–процедуры делятся на несколько категорий [19].

Параллельная графика (Parallel X graphics**).** Эти процедуры обеспечивают доступ всем процессам к разделяемому X–дисплею. Они создают удобный вывод для параллельных программ, позволяют чертить текст, прямоугольники, круги, линии и так далее.

Регистрация (Logging**).** Одним из наиболее распространенных средств для анализа характеристик параллельных программ является файл трассы отмеченных во времени событий – **логфайл** (logfile). Библиотека MPE создает возможность легко получить такой файл в каждом процессе и собрать их вместе по окончанию работы. Она также автоматически обрабатывает рассогласование и дрейф часов на множественных процессорах, если система не обеспечивает синхронизацию часов. Это библиотека для пользователя, который желает создать свои собственные события и программные состояния.

Последовательные секции (Sequential Sections**).** Иногда секция кода, которая выполняется на ряде процессов, должна быть выполнена только по одному процессу за раз в порядке номеров этих процессов. MPE имеет функции для такой работы.

Обработка ошибок (Error Handling). MPI имеет механизм, который позволяет пользователю управлять реакцией реализации на ошибки времени исполнения, включая возможность создать свой собственный обработчик ошибок.

Далее будут преимущественно рассмотрены средства регистрации (logging) для анализа эффективности параллельных программ. Анализ результатов регистрации производится после выполнения вычислений. Средства регистрации и анализа включают ряд профилирующих библиотек, утилитных программ и ряд графических средств.

Первая группа средств − профилирование. Библиотечные ключи обеспечивают собрание процедур, которые создают логфайлы. Эти логфайлы могут быть созданы вручную путем размещения в программе MPI обращений к MPE, или автоматически при

установлении связи с соответствующими MPE–библиотеками, или комбинацией этих двух методов. В настоящее время MPE предлагает следующие три профилирующие библиотеки:

1. **Tracing Library** (библиотека трассирования) – трассирует все MPI–вызовы. Каждый вызов предваряется строкой, которая содержит номер вызывающего процесса в **MPI_COMM_WORLD** и сопровождается другой строкой, указывающей, что вызов завершился. Большинство процедур **send** и **receive** также указывают значение **count**, **tag** и имена процессов, которые посылают или принимают данные.
2. **Animation Libraries (**анимационная библиотека**)** – простая форма программной анимации в реальном времени, которая требует процедур X–окна.
3. **Logging Libraries** (библиотека регистрации) – самые полезные и широко используемые профилирующие библиотеки в MPE. Они формируют базис для генерации логфайлов из пользовательских программ. Сейчас имеется три различных формата логфайлов, допустимых в MPE. По умолчанию используется формат CLOG. Он содержит совокупность событий с единым отметчиком времени. Формат ALOG больше не развивается и поддерживается для обеспечения совместимости с ранними программами. И наиболее мощным является формат – SLOG (для Scalable Logfile), который может быть конвертирован из уже имеющегося CLOG–файла или получен прямо из выполняемой программы (для этого необходимо установить пременную среды MPE_LOG_FORMAT в SLOG).

Набор утилитных программ в MPE включает конверторы лог-форматов (например, clog2slog), печать логфайлов (slog_print), оболочки средств визуализации логфайлов, которые выбирают корректные графические средства для представления логфайлов в соответствии с их расширениями.

Далее будут рассмотрены только библиотеки регистрации **Logging Libraries**. Результаты расчета времени дают некоторое представление об эффективности программы. Но в большинстве случаев нужно подробно узнать, какова была последовательность событий, сколько времени было потрачено на каждую стадию вычисления и сколько времени занимает каждая отдельная операция передачи. Чтобы облегчить их восприятие, нужно представить их в графической форме.

Но для этого сначала нужно создать файлы событий со связанными временными отметками, затем исследовать их после окончания программы и только затем интерпретировать их графически на рабочей станции. Такие файлы ранее уже названы логфайлами. Способность автоматически генерировать логфайлы является важной компонентой всех средств для анализа эффективности параллельных программ.

Далее в этой главе будут описаны некоторые простые инструментальные средства для создания логфайлов и их просмотра. Библиотека для создания логфайлов отделена от библиотеки обмена сообщениями MPI. Просмотр логфайлов независим от их создания, и поэтому могут использоваться различные инструментальные средства. Библиотека для создания логфайлов MPE разработана таким образом, чтобы сосуществовать с любой MPI–реализацией и распространяется наряду с модельной версией MPI.

Чтобы создать файл регистрации, необходимо вызвать процедуру **MPE_Log_event**. Кроме того, каждый процесс должен вызвать процедуру **MPE_Init_log**, чтобы приготовиться к регистрации, и **MPE_Finish_log**, чтобы объединить файлы, сохраняемые локально при каждом процессе в единый логфайл. **MPE_Stop_log** используется, чтобы приостановить регистрацию, хотя таймер продолжает работать. **MPE_Start_log** возобновляет регистрацию.

Программист выбирает любые неотрицательные целые числа, желательные для типов событий; система не придает никаких частных значений типам событий. События рассматриваются как не имеющие продолжительность. Чтобы измерить продолжительность состояния программы, необходимо, чтобы пара событий отметила начало и окончание состояния. Состояние определяется процедурой **MPE_Describe_state**, которая описывает начало и окончание типов событий. Процедура **MPE_Describe_state** также добавляет название состояния и его цвет на графическом представлении. Соответствующая процедура **MPE_Describe_event** обеспечивает описание события каждого типа. Используя эти процедуры, приведем пример вычисления числа π. Для этого оснастим программу вычисления числа π соответствующими операторами. Важно, чтобы регистрация события не создавала больших накладных расходов. **MPE_Log_event** хранит небольшое количество информации в быстрой памяти. Во время выполнения **MPE_Log_event** эти буфера

объединяются параллельно и конечный буфер, отсортированный по временным меткам, записывается процессом 0.

```
#include "mpi.h"
#include "mpe.h"
#include <math.h>
#include <stdio.h>

int main(int argc, char *argv[ ])
{  int  n, myid, numprocs;
   double PI25DT = 3.14159265358979323846264 3;
   double mypi, pi, h, sum, x, startwtime = 0.0, endwtime;
   int  event1a, event1b, event2a, event2b,event3a, event3b, event4a, event4b;
   char processor_name[MPI_MAX_PROCESSOR_NAME];
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPE_Init_log();
/*  Пользователь не дает имена событиям, он получает их из MPE */
/*   определяем  8 событий  для 4 состояний Bcast",”Compute",”Reduce" ,”Sync" */
   event1a = MPE_Log_get_event_number();
   event1b = MPE_Log_get_event_number();
   event2a = MPE_Log_get_event_number();
   event2b = MPE_Log_get_event_number();
   event3a = MPE_Log_get_event_number();
   event3b = MPE_Log_get_event_number();
   event4a = MPE_Log_get_event_number();
   event4b = MPE_Log_get_event_number();
   if (myid == 0) {
/* задаем  состояние "Bcast" как время между событиями event1a и event1b. */
        MPE_Describe_state(event1a, event1b, "Broadcast", "red");
/* задаем  состояние "Compute" как время между событиями event2a и vent2b. */
        MPE_Describe_state(event2a, event2b, "Compute",   "blue");
/* задаем состояние "Reduce" как время между событиями event3a и event3b. */
        MPE_Describe_state(event3a, event3b, "Reduce",    "green");
/* задаем состояние "Sync" как время между событиями event4a и event4b. */
        MPE_Describe_state(event4a, event4b, "Sync",   "orange");
   }
   if (myid == 0)
   {     n = 1000000;
        startwtime = MPI_Wtime();
    }
    MPI_Barrier(MPI_COMM_WORLD);
    MPE_Start_log();
                    /* регистрируем событие event1a */
    MPE_Log_event(event1a, 0, "start broadcast");
```

```
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
                    /* регистрируем событие event1b */
    MPE_Log_event(event1b, 0, "end broadcast");
                    /* регистрируем событие event4a */
    MPE_Log_event(event4a,0,"Start Sync");
    MPI_Barrier(MPI_COMM_WORLD);
                   /* регистрируем событие event4b */
    MPE_Log_event(event4b,0,"End Sync");
                   /* регистрируем событие event2a */
    MPE_Log_event(event2a, 0, "start compute");
    h   = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs)
    {  x = h * ((double)i - 0.5);
       sum += (4.0 / (1.0 + x*x));
     }
     mypi = h * sum;
                     /* регистрируем событие event2b */
     MPE_Log_event(event2b, 0, "end compute");
                       /* регистрируем событие event3a */
     MPE_Log_event(event3a, 0, "start reduce");
     MPI_Reduce(&mypi,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
                       /* регистрируем событие event3b */
     MPE_Log_event(event3b, 0, "end reduce");
     MPE_Finish_log("cpilog");
     if (myid == 0)
     {    endwtime = MPI_Wtime();
          printf("pi is approximately %.16f, Error is %.16f\n",
                 pi, fabs(pi - PI25DT));
          printf("wall clock time = %f\n", endwtime-startwtime);
     }
    MPI_Finalize();
    return(0);
}
```

Важный вопрос – доверие к локальным часам. На некоторых параллельных компьютерах часы синхронизированы, но на других – только приблизительно. В локальных сетях рабочих станций ситуация намного хуже, и генераторы тактовых импульсов в разных компьютерах иногда «плывут» один относительно другого.

Анализ логфайлов. После выполнения программы **MPI**, которая содержит процедуры MPE для регистрации событий, директорий, где она выполнялась, содержит файл событий, отсортированных по времени, причем время скорректировано с учетом «плавания» частоты

генераторов. Можно написать много программ для анализа этого файла и представления информации.

Например, одна из реализаций MPE содержит короткую программу, называемую **states**. Если мы выполняем ее с логфайлом, который описали выше, мы получим:

Состояние:	**Время:**
Broadcast	0.000184
Compute	4.980584
Reduce	0.000248
Sync	0.000095
Сумма:	**4.981111**

Такая итоговая информация является довольно распространенной, но грубой формой профилирования; она сообщает только, где программа тратит время. Значительно информативнее графическое представление, обеспечиваемое специализированными программами, например, **upshot** и **jampshot**.

Входные процедуры MPE используются, чтобы создать логфайлы (отчеты) о событиях, которые имели место при выполнении параллельной программы. Форматы этих процедур представлены ниже.

Форматы для C

int **MPE_Init_log** (void)
int **MPE_Start_log** (void)
int **MPE_Stop_log** (void)
int **MPE_Finish_log** (char *logfilename)
int **MPE_Describe_state** (int start, int end, char *name, char *color)
int **MPE_Describe_event** (int event, char *name)
int **MPE_Log_event** (int event, int intdata char *chardata)

Форматы для Fortran

MPE_INIT_LOG( )
MPE_FINISH_LOG (LOGFILENAME)
CHARACTER*(*) LOGFILENAME
MPE_START_LOG ( )
MPE_STOP_LOG ( )
MPE-DESCRIBE_STATE(START**,** END, NAME, COLOR)
INTEGER START, END
CHARACTER*(*) NAME, COLOR
MPE_DESCRIBE_EVENT(EVENT, NAME)
INTEGER EVENT

CHARACTER*(*) NAME
MPE_LOG_EVENT(EVENT, INTDATA, CHARDATA)
INTEGER EVENT,
INTDATA CHARACTER*(*) CHARDATA

Эти процедуры позволяют пользователю включать только те события, которые ему интересны в данной программе. Базовые процедуры – **MPE_Init_log**, **MPE_Log_event** и **MPE_Finish_log**.

MPE_Init_log должна вызываться всеми процессами, чтобы инициализировать необходимые структуры данных. **MPE_Finish_log** собирает отчеты из всех процессов, объединяет их, выравнивает по общей шкале времени. Затем процесс с номером 0 в коммуникаторе **MPI_COMM_WORLD** записывает отчет в файл, имя которого указано в аргументе. Одиночное событие устанавливается процедурой **MPE_Log_event**, которая задает тип события (выбирает пользователь), целое число и строку для данных. Чтобы разместить логфайл, который будет нужен для анализа или для программы визуализации (подобной upshot), процедура **MPE_Describe_state** позволяет добавить события и описываемые состояния, указать точку старта и окончания для каждого состояния. При желании для визуализации отчета можно использовать цвет. **MPE_Stop_log** и **MPE_Start_log** предназначены для того, чтобы динамически включать и выключать создание отчета.

Эти процедуры используются в одной из профилирующих библиотек, поставляемых с дистрибутивом для автоматического запуска событий библиотечных вызовов MPI.

Две переменные среды **TMPDIR** и **MPE LOG FORMAT** нужны пользователю для установки некоторых параметров перед генерацией логфайлов.

MPE LOG FORMAT – определяет формат логфайла, полученного после исполнения приложения, связанного с MPE–библиотекой. **MPE LOG FORMAT** может принимать только значения **CLOG**, **SLOG** и **ALOG**. Когда **MPE LOG FORMAT** установлен в **NOT**, предполагается формат **CLOG**.

TMPDIR – описывает директорий, который используется как временная память для каждого процесса. По умолчанию, когда **TMPDIR** есть **NOT**, будет использоваться “**/tmp**”. Когда пользователю нужно получить очень длинный логфайл для очень длинной MPI–работы, пользователь должен убедиться, что **TMPDIR**

достаточно велик, чтобы хранить временный логфайл, который будет удален, если объединенный файл будет создан успешно.

2.4. СРЕДСТВА ПРОСМОТРА ЛОГФАЙЛОВ

Существует четыре графических средства визуализации, распространяемых вместе с MPE: upshot, nupshot, Jumpshot-2 и Jumpshot-3. Из этих четырех просмотрщиков логфайлов только три построены с помощью MPE. Это upshot, Jumpshot-2 и Jumpshot-3.

Upshot и Nupshot. Один из используемых инструментов называется upshot. Самый простой вид экрана Upshot показан на рис. 2.2. Результат представлен в виде полос, по одной полосе на каждый процесс. Каждая полоса состоит из участков разного цвета (для черно-белых мониторов цвет заменяется различными штриховками). Каждому цвету соответствует состояние процесса определенного типа. Внизу приведены отметки времени, начинающиеся от нуля. Такое представление позволяет визуально определить состояние любого процесса в любой момент времени.

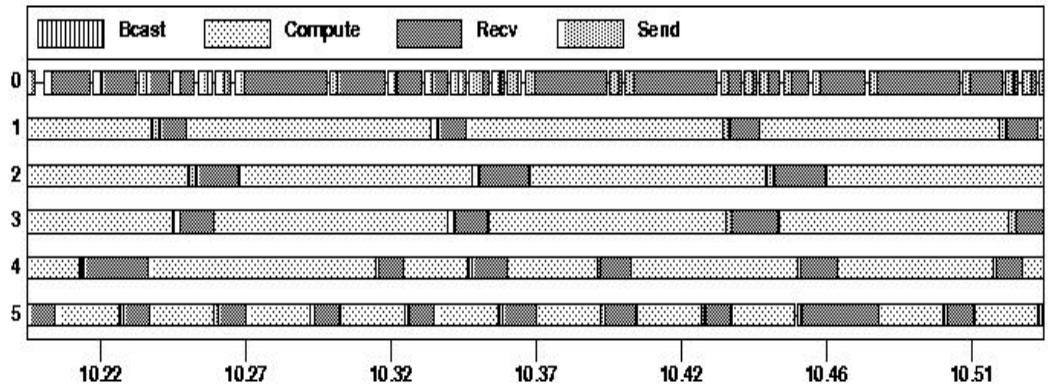

Рис. 2.2. Представление результатов профилирования с помощью upshot

Существуют и более сложные изображения окон, которые позволяют менять размер изображения по горизонтали или вертикали, центрировать на любой точке дисплея, выбранной мышью.

Jumpshot-2 и Jumpshot-3. Существует еще две версии, поставляемых вместе с MPE. Это Jumpshot-2 и Jumpshot-3, развившиеся из Upshot и Nupshot. Обе написаны на Java и являются графическими средствами визуализации для интерпретации двоичных файлов трасс, которые показывают их на дисплее.

Конкретные сведения по средствам просмотра обычно представлены в соответствующей реализации MPICH.