Ульяновский Государственный Университет

2000

Записка по курсовой работе

**“Менеджер управления распределенными вычислениями в локальной сети”**

Выполнил: студент группы ПМ-52

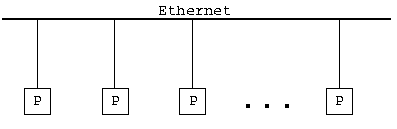
Никифоров Ю.В.

Преподаватель: Дулов Е.В.

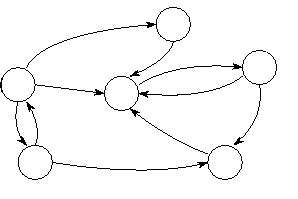
2000

# 1. Модель среды параллельного программирования

В качестве физической архитектуры параллельного компьютера используется локальная сеть LAN Ethernet. Таким образом, параллельный компьютер состоит из некоторого количества *процессоров* ***P***, соединенных между собой линией передачи данных.



В модели параллельного программирования используются две абстракции: *задача(task)* и *канал(channel)*.



Данная модель характеризуется следующими свойствами:

1. Параллельное вычисление состоит из одного или более одновременно исполняющихся задач (процессов), число которых может изменяться в течение времени выполнения программы.
2. Задача - это последовательная программа с локальными данными. Задача имеет входные и выходные порты, которые служат интерфейсом к среде процесса.
3. В дополнение к обычным операциям задача может выполнять следующие действия: послать сообщение через выходной порт, получить сообщение из входного порта, создать новый процесс и завершить процесс.
4. Посылающаяся операция асинхронная - она завершается сразу, не ожидая того, когда данные будут получены. Получающаяся операция синхронная: она блокирует процесс до момента поступления сообщения.
5. Пары из входного и выходного портов соединяются очередями сообщений, называемыми *каналами*. Каналы можно создавать и удалять. Ссылки на каналы (порты) можно включать в сообщения, так что связность может измениться динамически.
6. Процессы можно распределять по физическим процессорам произвольным способами, причем используемое отображение (распределение) не воздействует на семантику программы. В частности, множество процессов можно отобразить на одиночный процессор.

# 2. Временные характеристики параллельной программы

Время выполнения программы – время, прошедшее с момента запуска первого процессора до момента завершения выполнения последнего (получения результата).

***T = f (N, P, U, …)***

где *N* - размерность задачи, *P* - количество процессоров, *U* - количество задач параллельного алгоритма.

Во время выполнения каждый процессор может находиться в трёх состояниях: *вычисление (computation), обмен данными (communication) и ожидание (idle)*. Соответственно, определяется время нахождения процессора в каждом из них:

Следовательно, время выполнения ***T*** может быть определено следующим образом:



или



**Время вычисления алгоритма** *T*comp может быть равным времени выполнения соответствующего не распараллеленного (последовательного) алгоритма и зависит от размерности *N* задачи. Если параллельный алгоритм вносит дополнительные вычисления, тогда время вычисления зависит также и от количества задач *U* и процессоров *P*.

**Время обмена данными алгоритма** *T*comm это время, затраченное на прием и передачу данных между задачами. Существуют два вида обмена данными: *между процессорами и внутри процессора*. Первый тип обмена осуществляется между задачами находящимися на разных процессорах, т.е. по каналу связи. Второй тип обмена происходит, если взаимодействующие задачи находятся на одном процессоре, поэтому в данном случае обмен осуществляется гораздо быстрее, чем в первом, и по экспериментальным данным этим временем можно пренебречь.

Время передачи пакета данных между процессорами можно представить в виде следующего выражения:



где *t*s – время инициализации передачи, *t*w – время передачи единицы (слова) данных. Таким образом, в идеале имеем линейную зависимость времени передачи от длины данных.

Но в сети типа Ethernet для обмена данными для всех процессоров используется единственный канал связи. Если два процессора хотят передать данные в одно и то же время, реально будет передавать только один из них, а второй будет ожидать окончания передачи первого. Т.е. имеет место разделение канала связи во времени. Если *S* – количество конкурирующих процессоров нуждающихся в передаче данных, то предыдущая формула изменится следующим образом:



Таким образом, реальная пропускная способность канала равна *S*-1.

**Время ожидания (простоя) алгоритма** *T*idle. Процессор может простаивать в одном из двух случаев:

* при отсутствии загруженных на него задач;
* при отсутствии входных данных задачи.

Во втором случае избавится от простаивания можно следующим образом. Когда задача блокируется в ожидании входных данных можно на данном процессоре запустить другую задачу, для которой имеются входные данные. Как только для первой задачи поступят данные прекратить выполнение второй. Данный метод оправдан только при низкой стоимости переключения задач. Также можно подключать разные каналы для одной и той же задачи при низкой стоимости данной операции.

Более удобной мерой качества параллельной программы, чем время выполнения, является *эффективность*. Она характеризует полноту использования алгоритмом ресурсов параллельного вычислительной среды независимо от размерности самой задачи. Относительная эффективность определяется как:



где *T*1– время выполнения на одном процессоре, *T*p – время выполнения на *P* процессорах.

*Относительное ускорение:*



- это коэффициент уменьшения времени выполнения на P процессорах.

# 3. Методы измерения временных характеристик в реальном времени

Имеется три вида методов сбора информации о производительности параллельной программы:

* *рабочий профиль* программы (execution profile) представляет собой общее время, проведенное в различных участках программы;
* *счетчики* событий или совокупного времени;
* *трассировка событий*.

**Рабочий профиль** формируется автоматически для каждого процессора. При этом используется метод выборки данных через фиксированные промежутки времени, поэтому точность измерения не столь высока. По результатам можно построить гистограмму рабочих величин, например, определить задачу, которая забирает наибольшую часть вычислительного времени параллельного алгоритма на текущий момент.

**Счетчик** представляет собой переменную, которая может быть увеличена каждый раз при наступлении определенного события. Счетчик может использоваться для подсчета количества вызовов данной процедуры, общего количества переданных или принятых сообщений, общего размера переданных или принятых данных для каждого процессора, задачи или канала. Некоторые счетчики встроены в библиотеки среды параллельного программирования, и можно заводит новые путем вставки соответствующих вызовов библиотеки в исходный код задач.

Другой вариант счетчика – *интервальный таймер*. Он может использоваться для точного измерения времени выполнения определенных участков кода программы (задачи). Поэтому с помощью таймера можно измерять такой критичный ресурс процессора как производительность.

Информация, накопленная счетчиками, может использоваться в рабочих профилях программы.

**Трассировка** событий предоставляет наиболее детальную низкоуровневую информацию о производительности параллельной программы. Эта информация представляет собой записи с отметкой о времени наступления события, типом события, именем задачи, взаимодействующей задаче и др.

Трассировка событий может использоваться для локализации источников простоя и временной перегрузки программы, и так называемых «узких мест» в каналах передачи данных.

Полученные трассировочные данные могут также использоваться в рабочем профиле программы и счетчиках.

# 4. Реализация

**Метрики.** Измеряемые параметры производительности программы будем называть *метриками*, по сути, они те же счетчики. Каждая метрика представляет собой целое беззнаковое 32-битное число или *unsigned long*. Для каждого канала, задачи и процессора имеется стандартный внутренний набор метрик, который вычисляется автоматически или с участием программиста задач. Также имеется дополнительный массив ячеек для нестандартных метрик, размер его ограничен.

Ссылка на ячейки производится путем указания номера ячейки, аналогично массивам, начиная с нуля.

Для вычисления метрик используются три абстракции:

* *точки контроля* – это места вызова функций сбора данных о производительности (вход/выход процедуры и др.);
* *примитивы* – функции изменения значений метрик, запуска/останова таймеров;
* *предикаты* – условия вызова примитивов, основанные на метриках или локальных данных задачи.

Примитивы:

* установка счетчика в данное значение (setCounter);
* увеличение счетчика на заданную величину (addCounter);
* уменьшение счетчика на заданную величину (subCounter);
* установка таймера на данный счетчик (setTimer);
* запуск таймера (startTimer);
* останов таймера (stopTimer).

Пример: сколько времени данная функция проводит, посылая сообщения?

void foo ()

{

add (fooFlag); // fooFlag является признаком входа в функцию

. . .

sub (fooFlag);

}

sendMessage ()

{

if (fooFlag)

startTimer ();

. . .

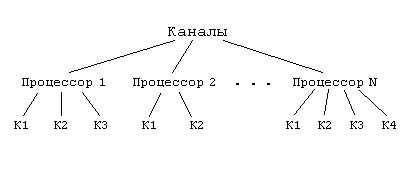
if (fooFlag)

stopTimer ();

}

**Ресурсы** – все интересующие нас объекты параллельной системы (процессоры, каналы, задачи). Ресурсы упорядочены в некоторое количество иерархий. В каждой иерархии представлены объекты определенного типа. На нижнем уровне иерархии находятся конкретные объекты данного типа, существующие на данный момент в системе. На следующем более высоком уровне они собираются по определенному признаку, например в объекты, содержащие их.

Пример иерархии каналов показан на рисунке.



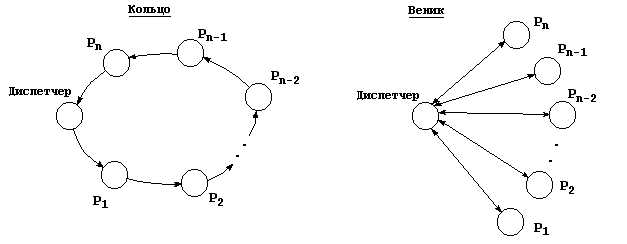
Иерархии ресурсов позволяют определить детализацию собираемой информации о производительности. Каждый более высокий уровень базируется на информации предоставляемой более низким уровнем. В примере с иерархией каналов, на самом нижнем уровне вычисляются метрики конкретных каналов. На уровне процессоров вычисляются общие метрики, такие как усредненные или суммарные значения метрик всех каналов на конкретном процессоре. На самом высоком уровне – всей системы – вычисляются общие метрики для всех каналов в системе.

Кроме иерархии каналов определены иерархии ресурсов процессоров и задач.

Иерархическая организация информации используется при анализе характеристик и производительности параллельных программ в реальном времени.

Для сбора информации используется метод *сэмплирования* (от английского sampling), т.е. периодического считывания текущих данных. Поступление очередного набора данных назовем *сэмплом* (sample).

**Кольцо служебных каналов.** Сбор информации о текущей производительности производится главным процессором (или задачей), называемой *диспетчером* или *менеджером*. Для этого используются служебные каналы, которые равноценны определенным ранее каналам. Из возможных структур сети служебных каналов, две из которых показаны на рисунке, выбрано “кольцо”.



При необходимости определенной информации диспетчер посылает запрос по служебному каналу, который не содержит данных. Каждый процессор, принимая запрос, дополняет его своей информацией и посылает дальше по служебному каналу, следующему процессору. В результате прохождения запроса по всем процессорам системы он возвращается к диспетчеру с накопленной информацией.

Целесообразность использования кольца служебных каналов (по сравнению с “веником”):

1. Небольшое общее количество односторонних каналов (N+1 вместо 2N);
2. Как следствие, разгрузка канала обмена диспетчера (нагрузка распределяется между всеми процессорами системы);
3. Более низкие требования к производительности диспетчера (есть время на обработку информации между соседними сэмплами);
4. Простота контроля загруженности канала сэмплирования (наряду со стандартными метриками каналов введены счетчики посланных и принятых запросов в диспетчере, в заголовок запроса включено поле – время отправления, по которому при возвращении запроса определяется время его осуществления);
5. Контроль потока служебной информации (легко регулируется периодичность запросов);
6. Простота сбора данных.

В каждом запросе указывается объект определенной иерархии и запрашиваемая метрика:

* вид ресурсов;
* уровень иерархии;
* номер объекта на данном уровне иерархии;
* код запрашиваемой метрики для данного объекта.

Структура получаемой информации однозначно определяется типом объектов указанных в запросе.

Возможно получение информации сразу по всем объектам и/или всем метрикам – при указании специальных значений последних двух полей. В данном случае в ответ на запрос возвращается массив однотипных структур.

**Диспетчер,** – выделенный процессор, управляющий и контролирующий параллельную систему, – осуществляет мониторинг, и сбор необходимой информации о системе в реальном времени. Определен прикладной программный интерфейс диспетчера, который может использоваться программами начального распределения задач, визуализации, анализа производительности, динамической оптимизации и др.

Таким образом, диспетчер поддерживает рабочий профиль параллельной программы. Общий вид структуры информации представляет собой двумерную матрицу. Одна размерность состоит из наименований однотипных объектов, другая – из наименований метрик, измеряемых для данных объектов. В качестве объектов используются процессоры, задачи и каналы. Таким образом, имеется три матрицы текущих параметров параллельной программы. Также, имеются вектора средних и общих параметров.

Для процессора вычисляются, например, следующие параметры: загруженность, количество памяти, время простоя и др.

Для задач вычисляются общее время вычисления, время обмена данными и др.

Для каналов: счетчики входов в процедуры обмена *send/recv*, объем переданных/принятых данных, среднее время нахождения в режиме приема/передачи и др.

# 5. Контроль производительности.

В данном разделе описываются идейные соображения для построения системы, анализирующей производительность параллельных программ в реальном времени.

Накопленная диспетчером информация – рабочий профиль – может использоваться для анализа выполнения параллельной программы. Далее описан примерный сценарий анализа.

Пусть задан вопрос: *работает ли программа эффективно*.

Гипотеза H0: программа работает эффективно.

Гипотеза H1: программа работает неэффективно.

Проверку гипотез производим из следующих соображений. Выделим основные типы неэффективной работы параллельной программы, один из них:

* большая доля времени простоя задач от общего времени работы.

Упрощенно это выражается следующим выражением:

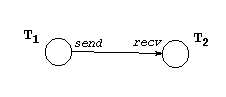


Можно взять критерий: *E*п *< 0,8.*

Итак, если время простоя занимает более 20 процентов общего времени работы программы, то есть основания считать работу программы неэффективной.

Подтвердив данную гипотезу – первого уровня, – переходим к анализу гипотез второго уровня. Предположим, в программе имеется «узкое место» в плане обмена данных. Рассмотрим участок графа потоков данных программы, показанный на рисунке.

Взаимодействуют две задачи через канал. Задача T1 генерирует данные, они передаются по каналу, являющемуся выходным для T1, а задача T2 принимает их на свой входной порт и обрабатывает.



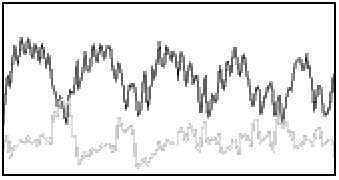
Существует два типа «узких мест»:

1. T2 не успевает обрабатывать входные данные, T1 находится в состоянии ожидания обработки выходных данных. При этом наблюдается:
2. загруженность T2 высока (> 90 %), загруженность T1 низка (< 50%),
3. количество сгенерированных L1 и обработанных L2 данных находятся в отношении L1 – L2 > d > 0,
4. длина очереди данных входного канала для T2 велика.
5. T1 генерирует мало данных, T2 находится в состоянии ожидания входных данных. При этом наблюдается:
6. загруженность T1 высока (> 90 %), загруженность T2 низка (< 50%),
7. длина очереди данных входного канала для T2 мала.

В данном случае можно выдвинуть две соответствующие гипотезы. Их проверку можно осуществить из следующих соображений.

Рассмотрим метрику канала связи процессоров – среднее время обмена. Для первой задачи это функция send, для второй – recv.

Пусть процесс изменения этих метрик во времени выглядят примерно так, как показано на рисунке.



Если имеет место первая причина снижения производительности, то первая метрика будет превышать вторую, т.к. первый процесс относительно долго находится в режиме передачи, а второй при этом не принимает данные, а, скорее всего, вычисляет.

Если имеет место вторая причина, то все наоборот.

Для подтверждения тех или иных гипотез можно применить методы анализа случайных процессов.

# Заключение

Описанная система параллельного программирования является основой для создания программ визуализации, анализа производительности, оптимального начального распределения задач, динамической оптимизации выполнения параллельных программ.

Программы визуализации и анализа производительности могут использоваться для изучения параллельных алгоритмов, как таковых.

В будущем, законченная система может использоваться для осуществления практических вычислительных задач большой сложности и оперирующих большими объемами данных.

Преимущество данной системы состоит в том, что она не требует применения мощных компьютерных систем, вместо этого она полноценно использует ресурсы любых локальных сетей на базе ОС Windows 95.

Литература:

1. Сервер ВЦ РАН http://www.ccas.ru/paral/
2. Ian Foster, “Designing and Building Parallel Programs”, 1995
3. Barton P. Miller and others, “The Paradyn Parallel Performance Measurement Tools” Computer Sciences Department, University of Wisconsin-Madison. http://www.cs.wisc.edu/paradyn/
4. Бертекас, Галлагер, “Сети передачи данных”, 1989.