электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.75

Система распределения вычислительной нагрузки на неоднородной вычислительной сети методом деревьев нитей

***Гусев А.П.**, студент*
кафедра «Компьютерные системы и сети»,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана

Научный руководитель: Руденко Ю.М., к.т.н., доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
v.suzev@bmstu.ru

В современном мире, со столь быстро развивающимися технологиями, идет постоянная технологическая гонка, в этом есть свои плюсы и минусы. Развитие техники всегда приносит научные плоды, например, на сверхмощных суперкомпьютерах проводят моделирование физических явлений и разрабатывают новые технологии, используются нами повсеместно, невозможно представить надежные немецкие автомобили, самолеты или столь сложную космическую технику, без компьютерного моделирования. Однако столь стремительное появление новинок на рынке технологий, заставляет многие организации тратить поистине невероятные суммы на покупку нового оборудования и переобучение персонала для работы с ним.

Как вам возможно известно, то же самое происходит и в сфере программного обеспечения, появляются новые подходы к реализации алгоритмов, новые технологии программирования и важнейшей проблемой является многопоточное программирование. Этот вопрос настолько сложен, что и по сей день не имеет максимально точного решения, так как эта задача не только программирования, но и математики.

Однако каждый из нас имеет в своем доме компьютер, или возможно даже несколько. В среднем внутри каждого из них находится многопоточный процессор, возьмем для примера, что каждый имеет по одному четырехядерному процессору. Это означает, что в вашем доме находится около 3200 процессоров (имеется ввиду многоквартирный панельный дом), можно представить, какой вычислительный потенциал они имеют (около 175 TFlop/s, что составляет 1% от вычислительной мощности суперкомпьютера лидирующего в топ-500)[2]. Зачастую, вся эта

вычислительная мощность не используется, но существуют способы максимально загрузить ресурсы с выгодой для каждого человека.

Для этого нами была разработана технология получившая название XMND (eXtreme Multi Node Divider). Для начала, рассмотрим общую структурную схему взаимодействия программ в нашей системе.

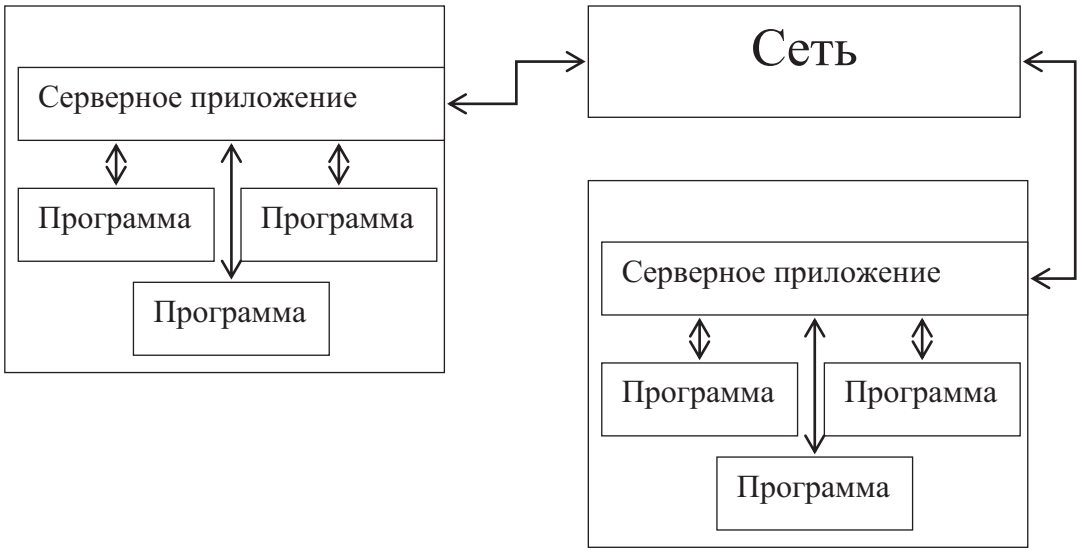

Рис. 1. Структурная схема взаимодействия

Как видно из схемы, каждое программное средство обращается к серверному приложению, которое регулирует распределение заданий по узлам сети. Такое общее представление весьма схоже с известной технологией MPI (Message Passing Interface)[8], но эта технология занимает более высокий уровень, и покрывает два главных недостатка MPI: Поддержку неоднородной архитектуры, устойчивость к ошибкам и распределение вычислительной нагрузки[8].

Чтобы произвести процедуру распределения, системе необходимо иметь исполняемый код и блок данных необходимых для обработки. Здесь и кроются главные проектные проблемы, первая состоит в необходимости передачи исполняемого кода, а вторая в передаче входных параметров и вычисленных значений.[1]

Рассмотрим ситуацию, когда на стартовой машине запускается приложение. Чтобы продемонстрировать работу системы, ниже на рисунке, приведен снимок исполняемого кода на языке C++.

Для начала приложение должно инициализироваться.[9] В понятие инициализации XMND входит создание сокетов и инициализация систем синхронизации, также система отправляет локальному узлу запрос на соединение с сервером вычислений. От сервера вычислений должен прийти идентификационный

номер, если в течение времени ожидания программа такового не получает, она переходит в режим самостоятельного выполнения.

```
#include "stdafx.h"
#include "controller.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int string_size = 0;
    int id = 0;
    initXMND(argc,argv);
    if(id = nareaXMND("nothing\0",0))
    {
        getXMND(id,"nothing\0",string_size);
        for (int i=0;i<100;i++)
        {
            if(id = nareaXMND("nothing\0",0))
            {
                getXMND(id,"nothing\0",string_size);
                //some code
                sendXMND(id,"nothing\0",0);
            }
            dividerXMND(id,"nothing\0",string_size);
        }
        sendXMND(id,"nothing\0",0);
    }
    dividerXMND(id,"nothing\0",string_size);
    releaseXMND();
    return 0;
}
```

Рис. 2. Пример использования в коде

Следующим шагом является выполнение секции помеченной пользователем, как вычисляемая посредством разделения ресурса.[3] Для организации обработки таких блоков была использована конструкция if, это временная мера, чтобы избежать внесения изменений в процесс компиляции программы.

Внутри функции nareaXMND, программа, прежде всего, проверяет режим вычислений: самостоятельный или с разделением. Если процесс происходит с разделением, программа посылает запрос к локальному серверу вычислений, для сверки своего идентификатор с тем, который приписан к блоку. Если он совпадает, программа переходит внутрь, если нет, блок ожидает своей очереди выполнения на сервере вычислений.

В качества параметра функции nareaXMND передается массив байтов и размер массива, он может быть принят внутри блока вычислений функцией getXMND через сервер вычислений.

Вычисленные данные в свою очередь передаются функцией sendXMND и могут быть приняты вызовом функции dividerXMND, которая принимает в качестве параметра идентификатор блока и будет ожидать его завершения в случае вызова.

При такой структуре приложение будет выполняться так называемым деревом нитей, схема процессов и их участков исполнения представлена на рисунке.

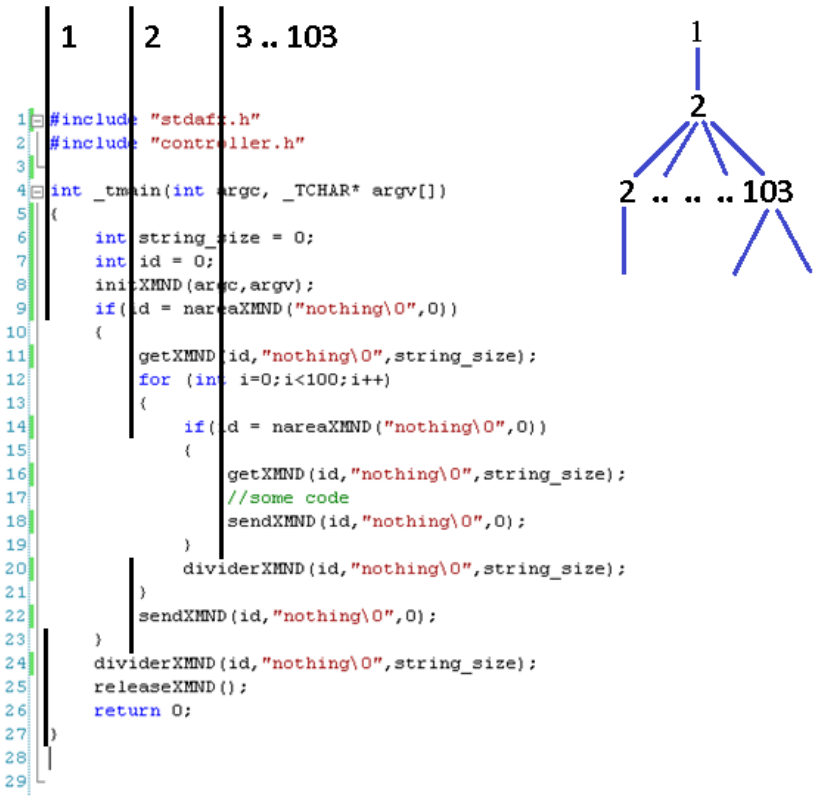

Рис. 3. Схема выполнения кода в процессах

Распределением нитей вычислений занимается уже не раз, упомянутый нами сервер вычислений. Для примера функционирования используемого алгоритма возьмем неоднородную вычислительную систему, представленную на рисунке 4.Б.

Модули сканируют определенный конфигурацией порт каждого компьютера в сети, и в случае обнаружения аналогичного модуля, сохраняют его в карте сети с средним временем полученным с помощью команды ping и значении скорости передачи данных, запуская соответствующий тест.

Следующим шагом каждого модуля является минимизация полученного дерева сети, с определением коммутационных групп и реконфигурации сети (Рис. 4.А), оставляя взаимодействие между группами узлам соответствующим условию:

max a*<производительность>i+b*<скорость сети>i+c*<интервал *ping*>i,

где a,b,c – коэффициенты задаваемые конфигурацией.

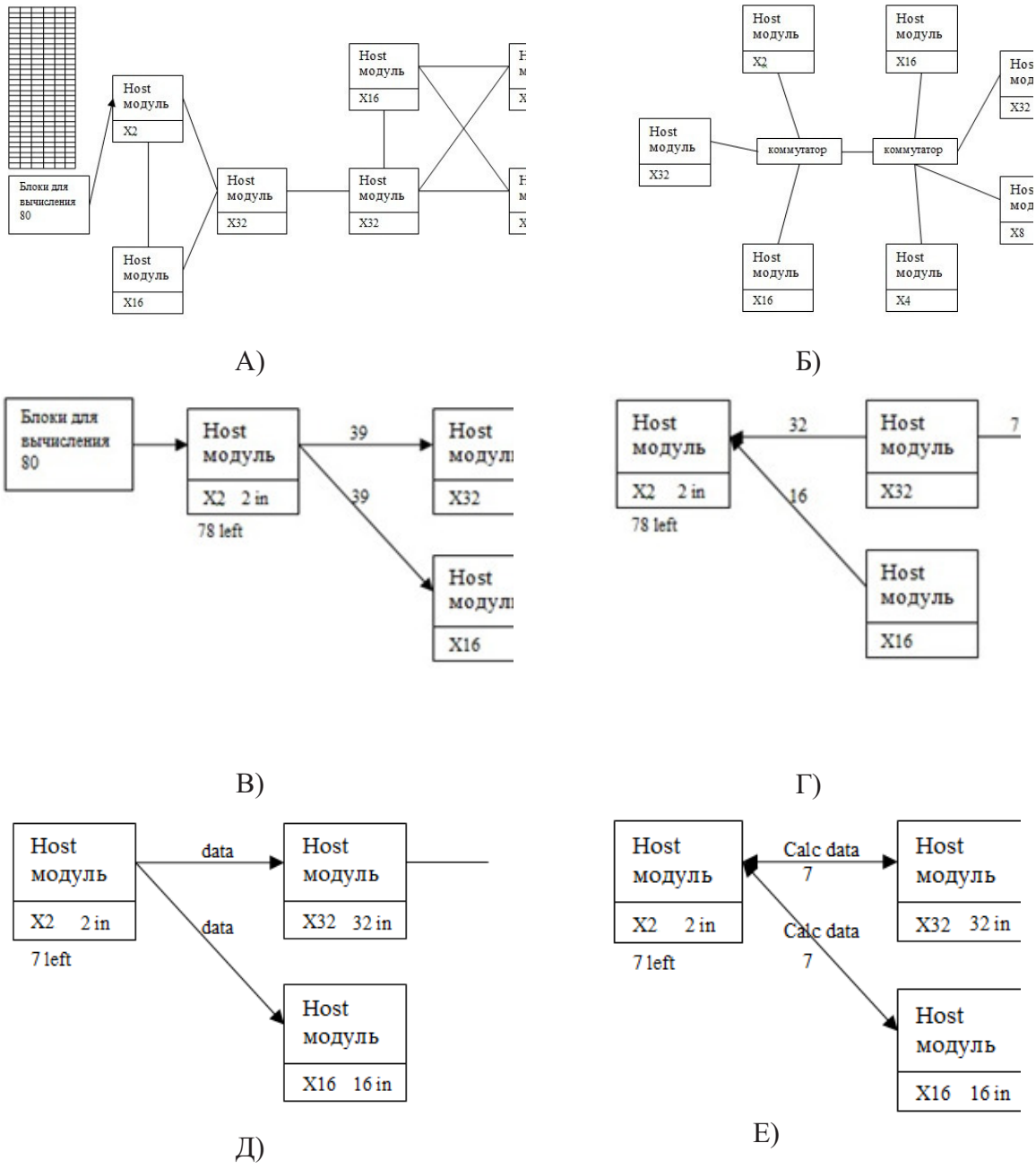

Рис. 4. Этапы работы предложенного алгоритма распределения

При поступлении на вход блока вычислений (80 блоков) *host*-модули начинают процедуру согласования распределения задач:

1. Резервирование блоков для самостоятельного вычисления
2. Рассылка запроса на вычисление оставшихся блоков по таблице вычислителей через промежутки времени (Рис 4.В).
3. Пересылка запрошенным вычислительным модулем согласия на вычисление, при наличии свободных мощностей и запроса на вычисление следующим вычислительным узлам (Рис 4.Г).
4. Получение согласия на вычисление и передача входных данных (Рис 4.Д).
5. Вычисление данных починенным узлом.

6. Пересылка вычисленных данных узлу-источнику, либо назначенному узлу (Рис 4.E).

7. Повторение операций.

Система, использующая данный алгоритм, имеет отклонение от максимальной производительности не превышающее 11%, которые теряются на задержки перед запуском процесса для передачи информации по сети.

В качестве завершения обзора разрабатываемой нами технологии, приведем таблицу запросов сервера и протокол его взаимодействия.

Команда	Параметры	Ответ	Описание
“S”	<S>< Порт><Тип вычислений><Данные исполняемого кода>	<Идентификатор процесса>	Запрос старта вычислений
“B”	<B><Общая оценка><Количество точных оценок><Массив точных оценок>		Маячок сетевых узлов
“H”	<H><Идентификатор области>	<Идентификатор процесса>	Запрос принадлежности области к процессу
“R”	<R><Порт><Количество блоков><Тип вычислений>< Идентификатор процесса ><Массив инициализации>	“A”	Запрос на вычисление блока
“A”	<A><Количество блоков><Идентификатор процесса>	“C” “D”	Подтверждение о старте вычислений
“C”	<C><Идентификатор процесса><Массив с данными кода>		Пересылка исходного кода
“D”	<D><Идентификатор процесса><Массив		Пересылка данных

	данных>		
“K”	<K><Идентификатор процесса><Массив вычисленных данных>		Пересылка вычисленных данных

Список литературы

1. B. Wilkinson and M. Allen. Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers, 2nd ed. Toronto, Canada: Pearson, 2004.
2. F. Gebali. Analysis of Computer and Communication Networks. New York: Springer, 2008.
3. T.G. Lewis and H. El - Rewini. Introduction to Parallel Computing. Englewood Cliffs, NJ: Prentice Hall, 1992.
4. B. Burke. NVIDIA CUDA technology dramatically advances the pace of scientific research. http://www.nvidia.com/object/io_1229516081227.html?_templated=320, 23.09.2012.
5. Cilk Arts. Smooth path to multicores. http://www.cilk.com/, 26.09.2012.
6. OpenMP. OpenMP: The OpenMP API specification for parallel programming. http://openmp.org/wp/, 2009.
7. C.E. Leiserson. The Cilk + + Concurrency Platform. Journal of Supercomputing, 51 (3), 2009.
8. MIP Forum. Message passing interface forum. http://www.mpi - forum.org/, 6.09.2012.
9. R.D. Blumofe and C.E. Leiserson. Scheduling multithreaded computations by work stealing. Journal of theACM (JACM), 46 (5), 1999.
10. OpenMP. Summary of OpenMP 3.0 c/c + + syntax. http://openmp.org/mp - documents/OpenMP3.0 -SummarySpec.pdf, 23.09.2012.
11. Maui Administrator's Guide. http://www.adaptivecomputing.com/resources/docs/maui/pdf/mauiadmin.pdf, 2.10.2012.
12. TORQUE Administrator Manual. http://www.clusterresources.com/torquedocs21/, 3.10.2012.