Министерство образования и науки
Российской федерации

Государственное образовательное учреждение
высшего профессионального образования
«Новосибирский государственный университет(НГУ)»

Механико-математический факультет
Кафедра вычислительных систем

Выпускная квалификационная работа бакалавра

ПУШКОВА Евгения Александровна

Разработка и реализация алгоритмов контроля исполнения фрагментированных программ

Научный руководитель
проф., д.т.н.
В.Э. Малышкин

Новосибирск 2011

Оглавление

ВВЕДЕНИЕ....3
Общая идея подхода фрагментированного программирования....3
Проблема отладки....5
Обзор....7
Анализ обзора....10
Постановка задачи....11
1. МОДЕЛЬ....14
1.1. Визуальное представление....17
1.2. Трассы и Профиль....17
1.3. Проблема несинхронности часов....18
1.4. Описание алгоритма корректировки времени....21
2. РЕАЛИЗАЦИЯ....24
2.1. Формат представления трасс и профиля....25
2.2. Конструктор трасс....27
2.3. Конструктор профиля....28
2.4. Визуализатор....30
ЗАКЛЮЧЕНИЕ....35
ЛИТЕРАТУРА....37

ВВЕДЕНИЕ

В настоящее время большое распространение получили суперкомпьютерные вычисления. В связи с этим возрастает необходимость повышения качества использования имеющихся вычислительных ресурсов. При этом возрастает требование к квалификации параллельного программиста в области системного параллельного программирования, то есть в области, вообще говоря, не связанной с прикладной задачей, которую требуется запрограммировать.

Поэтому существует необходимость создания и развития средств и технологий параллельного программирования, способных облегчить программирование суперкомпьютеров за счёт автоматизации системных составляющих параллельного программирования. В первую очередь, автоматически требуется обеспечить следующие прикладные свойства параллельной программы [1-4]:

- Настройка на имеющиеся аппаратные ресурсы;
- Динамическая балансировка нагрузки;
- Реализация коммуникаций на фоне вычислений;

Одной из таких технологий является технология фрагментированного программирования (ТФП) и система фрагментированного программирования LuNA [2-3], разрабатываемая в отделе Математического обеспечения высокопроизводительных вычислительных систем ИВМ и МГ СО РАН.

Общая идея подхода фрагментированного программирования

Подход ТФП состоит в том, чтобы пользователь задал описание алгоритма на высоком уровне, не связанном с конкретным аппаратным обеспечением, а конкретная параллельная программа, реализующая заданный алгоритм, сконструировалась системой программирования автоматически. Для повышения качества результирующей программы используются следующие основные приемы:

Агрегация – это объединение нескольких объектов (переменных и операций) в один неделимый с точки зрения системы программирования. Несколько операций/переменных рассматриваются как одна. Достоинством агрегации является уменьшение доли накладных расходов (т.е. доля системных ресурсов, в т.ч. времени) задействованных на организацию выполнения программы, а не на «полезные» вычисления. Недостаток – уменьшение возможного параллелизма за счет объединения данных и операций в неделимые блоки.

Задание рекомендаций по исполнению – это введение дополнительных ограничений на способ исполнения программы с целью отбросить заведомо неэффективные варианты её исполнения. Достоинством является более высокое качество исполнения программы. Недостатком является необходимость человека задумываться о способе исполнения программы на конкретном вычислителе.

Предметная область потенциального круга программируемых задач *ограничена* задачами численного моделирования. Это позволяет использовать специализированные алгоритмы исполнения параллельных программ, характерных для заданной предметной области, благодаря чему качество результирующих параллельных программ повышается. Кроме того, в области реализации больших численных моделей накоплен достаточно большой опыт, который может быть обобщён для решения задач этого круга.

В целом использование этих приёмов позволяет получать качество параллельной программы сравнимое с реализацией вручную [5-6].

Алгоритм в ТФП представляется в виде, называемом фрагментированной программой (ФП) [3]. ФП представляет собой набор из двух множеств:

- Множество *фрагментов данных* (ФД), где ФД - некоторая совокупность данных;
- Множество *фрагментов вычислений* (ФВ), где ФВ – некоторая процедура/функция, являющаяся «неделимой» с точки зрения параллелизма, то

есть должна быть выполнена последовательно. Входными и выходными переменными являются ФД;

Таким образом, пользователь может представить алгоритм в виде множеств ФВ и ФД, а затем, указав входные переменные, получить требуемые ФД, основываясь только на информационных зависимостями между ФВ (так называемое потоковое управление – с помощью каких ФВ можно получить значения одних ФД из других). Для повышения эффективности пользователь может задать указания по исполнению программы (прямое управление – порядок срабатывания ФВ, заданный отношением частичного порядка).

Далее, ФП компилируется и настраивается на конкретную конфигурацию мультикомпьютера (т.е. параллельного компьютера с распределённой памятью). Специальное окружение, называемое Исполнительной Системой, распределяет ФД на имеющиеся ресурсы и исполняет ФВ в соответствии с информационными зависимостями и рекомендациями пользователя [6-7].

Проблема отладки

Как и любой системе программирования, системе ФП требуется средство для поиска и отладки ошибок. Стоит отметить сложность отладки параллельных программ, существенно превышающую сложность отладки последовательных программ [8].

Обусловлено это необходимостью отлаживать не только функциональные, но и поведенческие свойства ФП: во-первых, требуется отслеживать работу программы на нескольких процессорах; во-вторых, от запуска к запуску фрагменты данных и вычислений могут отображаться на различные процессоры, коммуникации по аппаратным причинам происходить с различной задержкой и скоростью, операции синхронизации могут сработать в разном порядке и т.п. Все это является причиной недетерминизма исполнения параллельной программы. Таким образом, при тестировании возможна следующая ситуация – при многочисленном запуске программы на одних и тех же начальных/входных данных результаты будут правильными, и лишь спустя долгое время ошибка может проявить себя.

Следовательно, не имея должных отладочных средств, пользователь может потратить значительное количество времени на поиск и отладку ошибок.

Возможность определения прямого управления накладывает на пользователя ответственность за исполнение программы (так как ИС исполняет ФП в соответствии с указаниями пользователя). Поэтому особенностью отладки ФП является необходимость обнаруживать ошибки, совершенные самим пользователем. Например, возможны следующие ситуации:

а) **Потеря данных**.

Два ФВ **A** и **B**, имеющие в качестве выходного параметра один и тот же ФД **x**, могут сработать один сразу после другого, вследствие чего значение, вычисленное первым заместится другим значением.

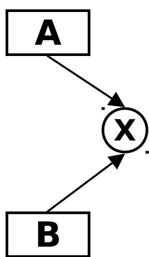


Рис. 0.1.

б) **Перехват данных**.

Предположим, что ФВ **B** требуются на вход данные, полученные вычислением ФВ **A**. Информационные зависимости позволяют ФВ (или блоку ФВ) **C** изменить данные, содержащиеся во ФД **x**.

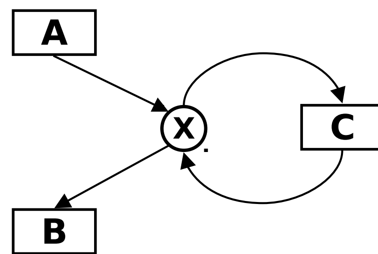


Рис. 0.2.

Возможно, такое управление и не является ошибкой, а задумано пользователем и ведет к выработке желаемого результата, но, как правило, такая ситуация говорит об ошибке и пользователю необходимо выдать предупреждение о подобных ситуациях.

Один из методов, позволяющих анализировать поведение программы с целью улучшения ее производительности: сбор информации о программе во время ее выполнения (трассировка).

Можно выделить два основных подхода:

A. "Трассировка + Визуализация":

A1. Во время исполнения программы собирается трасса - информация о ходе работы программы.

A2. Затем полученная трасса просматривается и анализируется.

B. "Онлайн-анализ". Поведение программы анализируется непосредственно в ходе ее выполнения.

Кроме того, трассировка позволяет определять количественные характеристики исполнения программ, что позволяет скорректировать распределение ресурсов. Задавая рекомендации о местонахождении тех или иных элементов, пользователь или система программирования способны оптимизировать исполнение программы.

Далее предоставлена информация о некоторых наиболее известных программных средствах, реализующих такие подходы.

Обзор

1. **AIMS - Automated Instrumentation and Monitoring System [9]**

Некоммерческий продукт типа A, разрабатывается в NASA Ames Research Center в рамках программы High Performance Computing and Communication Program.

Языки/Библиотеки: Fortran 77, HPF, С. Библиотеки передачи сообщений: MPI,PVM,NX.

Автоматизированное изменение исходного кода программы путем вставки специальных вызовов. Параллельно создается файл со статистической информацией. Уровни детализации - подпрограммы, вызовы процедур, процедуры различного типа (процедуры ввода-вывода, MPI процедуры т.п.).

Визуализация:

1. Процессы - параллельные линии. События изображаются точками на этих линиях. Особым образом изображаются накладные расходы: времена ожидания, блокировка. Есть возможность

"проигрывания" трасс, то есть имитация исполнения программы по сохраненному сценарию.

2. Время - реальное (астрономическое).

3. Связь линий процессов линиями, обозначающими взаимодействия (передача сообщений, глобальные операции).

4. Диаграммы взаимодействия процессов, временные срезы, история вызовов и трассируемых блоков.

5. Поддерживается связь с исходным кодом.

2. **Vampir, VampirTrace [10-12]**

Коммерческий продукт типа A (VampirTrace - система генерации трасс, Vampir - система визуализации), Center for Information Services and High Performance Computing (ZIH) of TU Dresden.

Языки/библиотеки: Fortran, C; передача сообщений в рамках MPI.

Функциональность трассировки: Слабые возможности настройки уровня детализации - только по подпрограммам. Возможна установка точек начала/конца трассировки.

Визуализация:

1. Процессы - параллельные линии, события - точки на них.

2. Взаимодействия. Связь линий процессов, матрицы объемов и количества пересылок.

3. Другие объекты: круговые диаграммы и статистические гистограммы.

4. Поддерживается связь с исходным кодом.

Статистика: Суммарное время по замеряемым инструкциям или типам инструкций и количеству срабатываний; отображается на круговых диаграммах и гистограммах.

3. **Jumpshot [13]**

Некоммерческое средство типа А2, разработано в Аргоннской национальной лаборатории. Распространяется вместе с пакетом MPICH. Jumpshot входит в состав MPICH начиная с версии 1.1.1 и заменяет собой Tcl/Tk-программы upshot/nupshot, входившие в состав MPICH более ранних версий.

Языки/библиотеки: Передача сообщений: MPI.

Функциональность трассировки: Для получения трассы программу необходимо откомпилировать с профилировочной версией библиотеки MPICH. Формат трасс - CLOG. Визуализация - Java.

Визуализация:

1. Процессы - параллельные линии, цветом изображается тип функции.
2. Взаимодействия. Связь линий процессов.
3. Другие объекты: объемы пересылок по времени, гистограммы накладных расходов по времени.

4. **CXperf (HP Performance Analysis Tools) [14]**

Коммерческое средство типа А, разработка Hewlett-Packard.

Языки/библиотеки: HP ANSI C (c89), ANSI C++ (aCC), Fortran 90 (f90), HP Parallel 32-bit Fortran 77.

Функциональность трассировки: Сбор и настройка трасс осуществляется с помощью указания специальных профилировочных опций компилятора.

Визуализация: 3D-визуализация, связь с кодом программы, масштабирование, сопоставительный анализ, графы вызовов.

Другие средства анализа поведения параллельных программ:

XMPI - графическая среда запуска и отладки MPI-программ, входит в состав пакета LAM;

HP Pak - набор средств от Hewlett-Packard для анализа поведения многопоточных программ;

TAU (Tuning and Analysis Utilities) - некоммерческий набор утилит анализа производительности программ, написанных на языке C++ и его параллельных вариантах. Включает пакет профилировки TAU Portable Profiling;

Carnival;

Chiron - средство для оценки производительности многопроцессорных систем с общей памятью;

Pangaea;

GUARD - параллельный отладчик;

MPP-Apprentice - средство в составе Message-Passing Toolkit от SGI;

ParaGraph;

PGPVM2;

TraceInvader;

XPVM - графическое средство мониторинга PVM-программ;

Pablo Performance Analysis Toolkit Software;

Paradyn;

Intel Thread Profiler;

Анализ обзора

Общие проблемы всех средств трассировки:

1. Формат трасс не унифицирован и обычно ориентирован на конкретную библиотеку передачи сообщений.
2. Сбор информации - слабые возможности настройки фильтров событий (какие события и какую информацию включать в трассы). Нет возможности варьировать объем трассы.
3. Не учитывается эффекта замера - средство трассировки достаточно сильно изменяет поведение программы.

В нашем случае, очевидно, подход B не подходит, так как решение больших задач требует длительного времени вычислений и, соответственно,

отлаживать программу онлайн в течение нескольких суток (или более) затруднительно.

Что касается средств, использующих подход А, то они в чистом виде неприменимы, т.к. направлены на анализ низкоуровневого программирования, т.е. собирается информация о передаче сообщений и блоках кода. При этом не учитывается фрагментированная структура программы.

Постановка задачи

Исходно алгоритм представляется рекурсивно-перечислимым множеством функциональных термов конечной длины. В реальности само множество термов не строится, а строится множество переменных и операций, а также задаётся множество входных и выходных переменных [1, 15].

В ТФП алгоритм фрагментируется. Таким образом, при агрегации атомарных термов соответственно происходит агрегация операций и переменных, формируются ФД и ФВ [3].

Интерпретация есть означивание всех переменных алгоритма конкретными значениями. Корректность интерпретации заключается в том, что при различных способах вычисления ФД из начальных данных, полученные значения должны совпадать. Таким образом, при корректной интерпретации, ошибки вида 1а не являются ошибками, так как ФВ **A** и **B** вычисляют одно и то же значении ФД **x** – и данные не теряются.

Важной особенностью задачи является то, что имеется описание алгоритма, с которым можно сравнивать конкретную его реализацию.

В целом, отладку ФП можно разделить на отладку самих ФВ, как последовательных подпрограмм (этот вопрос в работе не затрагивается), и на отладку исполнения ФВ (поведенческий отладчик [8]), считая ФВ отлаженными. При отладке исполнения основной задачей является проверка соответствия реализованного в ходе выполнения ФП порядка вычислений задуманному.

Первый вариант – это наглядно визуализировать реализованный порядок выполнения ФВ для визуального контроля человеком (напр., [16])

Второй вариант – анализировать профиль исполнения программы на основе описания алгоритма.

Это определяет актуальность проблемы.

Помимо ошибок пользователя существуют следующие задачи:

- проверка допустимости порядка выполнения операций (соответствие заданному порядку в принципе);
- проверка реализации заданных термов;
- проверка корректности интерпретации;
- поиск неиспользуемых значений - т.е. тех, которые вычисляются, но не используются;

В работе не ставится цель проверки корректности интерпретации, но поднимается вопрос поиска и предоставления информации о местах вероятного нарушения корректности.

Цель работы: разработать и реализовать алгоритмы профилирования исполнения ФП и визуализации полученных данных в наглядном виде для поиска и отладки ошибок программиста.

К реализации предъявляются следующие требования.

- *Переносимость*. Средство профилирования и визуализация должно работать на любой системе, где есть MPI.
- *Наглядность визуализации*. Поиск указанных ошибок должен быть прост и удобен. Визуализация должна наглядно для человека представлять информацию о порядке выполнения ФВ, чтобы он мог выявлять порядок выполнения ФВ, несоответствующий желаемому.
- *Малая доля накладных расходов*. Время сбора и обработки информации об исполнении программы должно быть мало по сравнению со временем исполнения самой программы («эффект наблюдателя» должен быть мал).
- *Профилирование*. Необходимо фиксировать действия, производимые с ФД, а также продолжительность и порядок срабатывания ФВ –

все это направлено на определение количественных показателей характеристик исполнения ФП и её частей, а также загрузки вычислительных узлов и коммуникационной подсистемы.

Научная новизна данной работы состоит в разработке:

1. Визуального представления исполнения ФП.
2. Алгоритмов профилирования исполнения ФП.
3. Алгоритмов обработки и визуализации информации об исполнении ФП.
4. Алгоритмов контроля исполнения ФП.

1. МОДЕЛЬ

Определим формально ФП . ФП – это:

- множество Фрагментов Данных (ФД), где ФД - некоторая совокупность данных. ФД представляют переменные алгоритма и являются агрегатами переменных. ФД – множественного присваивания аналогично переменным в императивных языках программирования. Это означает, что в разные моменты времени ФД может хранить, вообще говоря, различные значения. Как правило, ФД реализуется блоком памяти. Часть значений ФД перед исполнением ФП может быть инициализирована пользователем. Такие ФД называются входными ФД.
- множество Фрагментов Вычислений (ФВ), ФВ задаёт операцию над ФД, для него определены входные и выходные ФД. ФВ вычисляет значения выходных ФД из входных. Это называет срабатыванием ФВ. Каждый ФВ срабатывает только один раз за время исполнения ФП. Как правило, ФВ реализуется некоторой последовательной процедурой/функцией.
- На множестве ФВ задано отношение частичного порядка <, которое задаёт зависимости по срабатыванию ФВ: то есть если ФВ А < ФВ В, то, значит, А должен сработать раньше В. Собственно исполнение ФП заключается в срабатывании всех ФВ в соответствии с заданным порядком. Т.к. отношение < - это отношение частичного порядка, то допустимы различные варианты исполнения ФП, но при этом порядок < не должен нарушаться. В частности, возможно параллельное исполнение ФВ, если между ними нет зависимости (они несравнимы в смысле <).

Рассмотрим пример фрагментированной программы, реализующей некоторую одномерную разностную схему.

Имеется одномерная регулярная сетка значений x_i^t, где t – номер временного слоя. Значения каждого узла сетки на t+1 временном слое вычисляется из значений x_{i-1}^t, x_i^t и x_{i+1}^t (рис. 1.1.). Сетка разрезается на N блоков

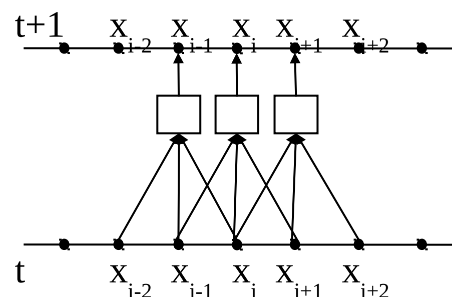


Рис. 1.1.

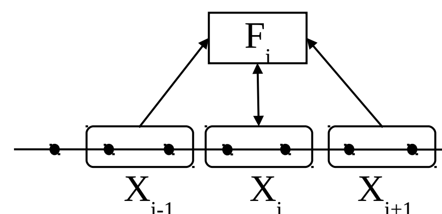


Рис. 1.2.

равного размера (случай, когда размер сетки не делится нацело на N рассматривать не будем для простоты изложения). Каждый блок реализуется одним ФД X_i, где i=1,…,N. На множестве ФД определяются ФВ F_i^t, вычисляющие значения переменных на следующем временном слое. Входными ФД для F_i^t будут X_{i-1}, X_i, X_{i+1}, а выходным – X_{i+1}. Таким образом, значения переменных для следующего временного слоя сохраняются в те же ФД (рис. 1.2.). Для корректной работы программы необходимо определить управление, т.е. задать отношение < на множестве ФВ. Оно задаётся следующим образом: $F_i^{t+1} > \{F_{i-1}^t, F_i^t, F_{i+1}^t\}$.

Таким образом, ФП будет состоять из следующих частей:

ФВ: $\{ F_i^t \}$

ФД: $\{ X_i \}$

Out(F_i^t) = X_i In(F_i^t) = $\{ X_{i-1}, X_i, X_{i+1} \}$

Частичный порядок: $F_i^{t+1} > \{F_{i-1}^t, F_i^t, F_{i+1}^t\}$

где i=1…N, t=0…T (T – количество итераций)

Входными ФД в этой ФП будут X_i.

При исполнении ФП в мультикомпьютере ФД и ФВ распределяются по вычислительным узлам мультикомпьютера. Чтобы ФВ мог сработать, все его входные и выходные ФД должны находиться на том же вычислительном узле. Как следствие, ФД мигрируют с узла на узел в процессе выполнения ФП. Кроме этого, для обеспечения динамической балансировки загрузки вычислительных узлов могут мигрировать и ФВ, таким образом, перераспределяя вычислительную загрузку. Динамическая балансировка нагрузки на процессоры производится с целью обеспечения равномерной загрузки вычислительных узлов для ускорения вычислений [6].

Рассмотрим исполнение ФП на мультикомпьютере на рассмотренном выше примере явной конечно-разностной схемы. ФД распределяются по узлам примерно поровну. Например, при исполнении ФП на двух узлах возможное распределение показано на рис. 1.3. пунктирной линией. Такое распределение оптимально в смысле перекрёстных (межузловых) связей, благодаря чему экономится нагрузка на сетевую подсистему мультикомпьютера.

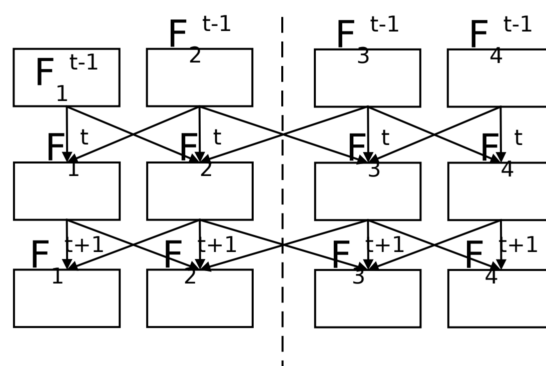


Рис. 1.3.

Входные ФД распределяются по узлам мультикомпьютера. Далее исполнение происходит следующим образом: ФВ срабатывают как только входные ФД данного ФВ находятся на одном узле и заданное управление позволяет исполнение (все ФВ, от которых зависит текущий ФВ, уже сработали).

Исчерпывающую информацию об исполнении ФП с точки зрения предъявляемых требований, дают следующие данные:

- Время начала и конца исполнения ФВ;
- Место исполнения ФВ, то есть на каком узле тот сработал;
- Время и траектория движения ФД по вычислительным узлам;
- Последовательность использования фрагмента данных фрагментами вычислений – порядок изменения содержимого фрагмента данных фрагментами вычислений;

Помимо поиска ошибок, данная информация о программе используется для оптимизации исполнения и компиляции. Например, данные о загрузке узлов мультикомпьютера во времени и о времени исполнения ФВ и передачи ФД могут быть использованы для более равномерного и полного использования аппаратных ресурсов.

1.1. Визуальное представление

Исполнение ФП визуализируется в виде *схемы-графика*, изображающего схему мультикомпьютера во времени с обозначенными срабатываниями ФВ и траекториями движения ФД по узлам (рис. 1.4.).

Для каждого узла изображается отдельная временная ось $\mathbf{t_i}$, на которой отображаются события, произошедшие на этом узле. ФВ обозначаются прямоугольниками, начало и конец которых обозначают соответственно время начала и конца исполнения ФВ. Коммуникации между узлами (миграция ФД) изображаются стрелками, соединяющими на различных осях точки, соответствующие временным моментам отправки и получения сообщения.

Нулем исполнения некоторого узла будем называть зафиксированное время начала исполнения программы на этом узле.

Глобальной осью назовем некоторую временную ось **T**, относительно которой будет рассматривать оси узлов. Соответственно время глобальной оси называется глобальным, а время узлов – локальным.

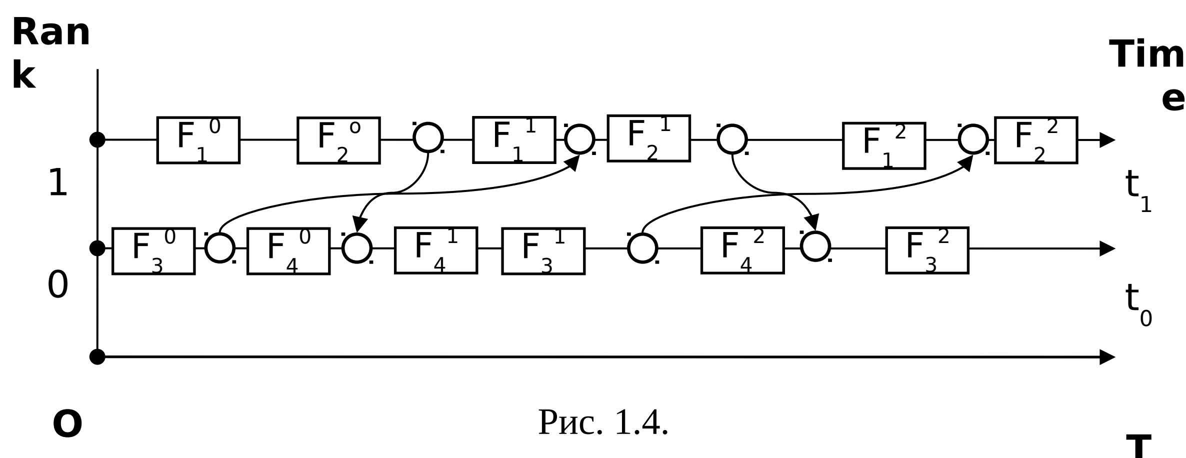


Рис. 1.4.

1.2. Трассы и Профиль

Трассой будем называть последовательность событий, зафиксированных на одном узле мультикомьютера. Событиями считаются: создание, удаление, посылка и прием ФД, начало и конец исполнения ФВ. В визуальном представлении события трассы изображаются на временной оси своего узла.

Трассы со всех узлов объединяются в один объект, назовём его *профилем*. Профиль – это объединение множества событий со всех трасс с заданным

линейным порядком на этом объединении. Другими словами, профиль есть проекция событий осей узлов на глобальную ось. Кроме того, профиль хранит дополнительные виды событий начала и завершения использования ФД ФВ.

Заметим, что события начала и конца использования ФД ФВ прямому отображению не подлежат. Тем не менее, в профиле они фиксируются для облегчения поиска пути ФД по ФВ.

Схематично профиль можно представить в виде следующей таблицы, где события представлены в виде последовательности строк вида: время, номер узла, суть события (его тип, описание):

Time	**Rank**	**Operation**
00:00:00	<rank>	receive <df id> from -1
…		
00:00:04	<rank1>	send <df id> to <rank2>
…		
00:02:45	<rank>	execution starts <cf id>
…		
01:32:00	<rank5>	receive <df id> from <rank5>
…		
02:15:34	<rank>	execution ends <cf id>
…		
03:34:56	<rank2>	receive <df id> from <rank1>
…		
13:15:00	<rank>	delete <df id>
etc…		

1.3. Проблема несинхронности часов

Если проецировать трассы на глобальную ось путем отождествления нулей исполнения с глобальным нулем O (то есть просто упорядочить все события во всех трассах по возрастанию времен – и получить профиль), то из-за возможной несинхронности часов на вычислительных узлах нельзя гарантировать сохранение реального порядка событий.

Под несинхронностью понимается несоответствие показаний часов на различных узлах мультикомпьютера. В результате этой несинхронности

зафиксированное время события может быть больше времени другого события, в действительности произошедшего ранее, т.е. в профиле нарушится реальный порядок событий.

Таким образом, просто объединить все трассы и упорядочить события по возрастанию времен недостаточно для получения профиля. Для решения этой проблемы предложен алгоритм, корректирующий зафиксированные локальные времена событий.

Рассмотрим сначала несоответствия времен отправки/получения ФД (если они есть): время отправления ФД превышает время получения на некоторую величину delta>0. Назовем такую ситуацию *конфликтом*.

Профиль, не содержащий конфликтов, будем называть *корректным.*

При обнаружении конфликта, ноль исполнения для узла-получателя сдвигается в большую сторону относительно глобальной оси на величину delta. Вследствие этого конфликт устраняется. В результате ликвидации такого вида неточностей будет сконструирован корректный профиль, если часы идут ровно и погрешность измерения достаточно мала. Докажем это.

Для простоты рассмотрим 2 узла мультикомпьютера. Назовем их первым и вторым. Есть коммуникации от первого ко второму и от второго к первому.

Индукция по числу конфликтов. При m=0 очевидно. Пусть просмотрены m конфликтов и при помощи сдвигов они успешно устранены.

Предположим, что для устранения (m+1)-го конфликта необходимо сдвинуть ноль первого узла на величину delta = $u - v$, где u и v - соответственно глобальные времена получения и отправки ФД, c – действительное время пересылки. На рис. 1.5. и 1.6. представлены части схемы исполнения ФП соответственно до коррекции и после. Для наглядности изображены только коммуникации.

Время на передачу сообщений (разница времен отправки и получения) от второго к первому увеличилось на delta. Поэтому очевидно, что новый конфликт имеет место в некоторой коммуникации от первого узла ко второму.

Предположим, что после сдвига ($v' = v$ + delta, $v' = u$) образовался новый конфликт $g' > f$, где g' – скорректированное время отправления некоторого фрагмента данных DF (g - до коррекции, $g \leq f \rightarrow w = f - g <$ delta), f – время его получения. Время затраченное на пересылку DF в действительности – a.

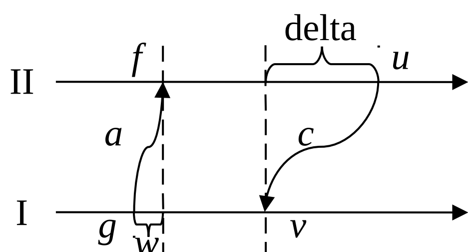


Рис. 1.5.

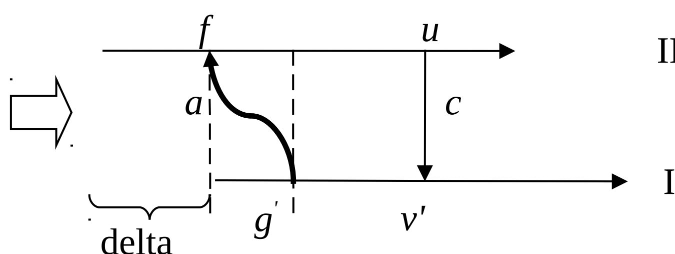


Рис. 1.6.

Очевидно, u-f должно быть меньше или равно v'-g', т.к. в действительности ФД пребывал на узле II после того как был отправлен с узла I и до того, как был получен на узле I. Однако, из предположения следует, что время между v' и g' меньше чем зафиксированное время между f и u (иначе конфликта бы не существовало, $u = v'$).

$v' - g' = a + u - f + c$

$g' = g +$ delta, $v' = u \rightarrow - g -$ delta $= a + c - f \rightarrow$

$\rightarrow w = f - g = a + c +$ delta $\rightarrow$ delta $< w$

Противоречие. Следовательно, предлагаемый алгоритм за конечное число шагов вырабатывает корректный профиль, что и требовалось доказать.

В идеале, нам хотелось бы восстановить реальную картину исполнения программы: в точности знать что когда произошло. Но получение такой картины крайне затруднено в силу погрешностей измерений и несинхронностей часов и является отдельной проблемой. Тем не менее, корректного профиля вполне достаточно для наших целей, так как нам важно показать не столько абсолютные показатели времени передачи ФД, сколько последовательность срабатываний ФВ относительно общих для них ФД. Важно знать кто когда и за кем изменил ФД: если ФВ не связаны информационной зависимостью, то порядок их исполнения не является существенным для отладки. Порядок же

ФВ, сработавших на разных узлах и связанных информационной зависимостью, после коррекции будет соответствовать реальному.

Описанную проблему можно проиллюстрировать на примере программы, реализующей разностную схему (пример которой приведен в начале данной главы). Обратим внимание, что на примере (рис. 1.7.) время первого получения данных узлом 0 меньше, чем время отправки их узлом 1 ($y<x$). Это означает, что нужно сдвинуть ось узла 0 так как показано на рис 1.8.

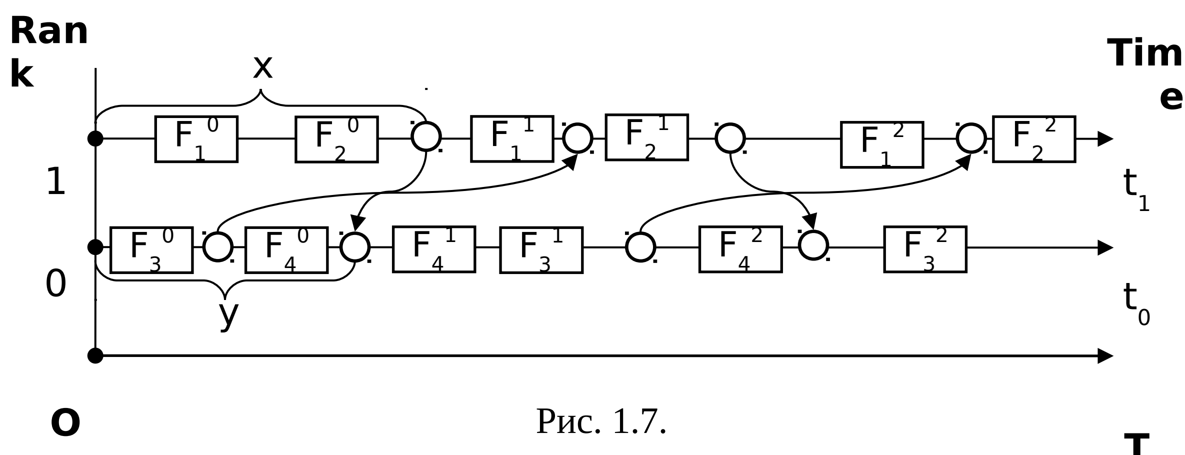


Рис. 1.7.

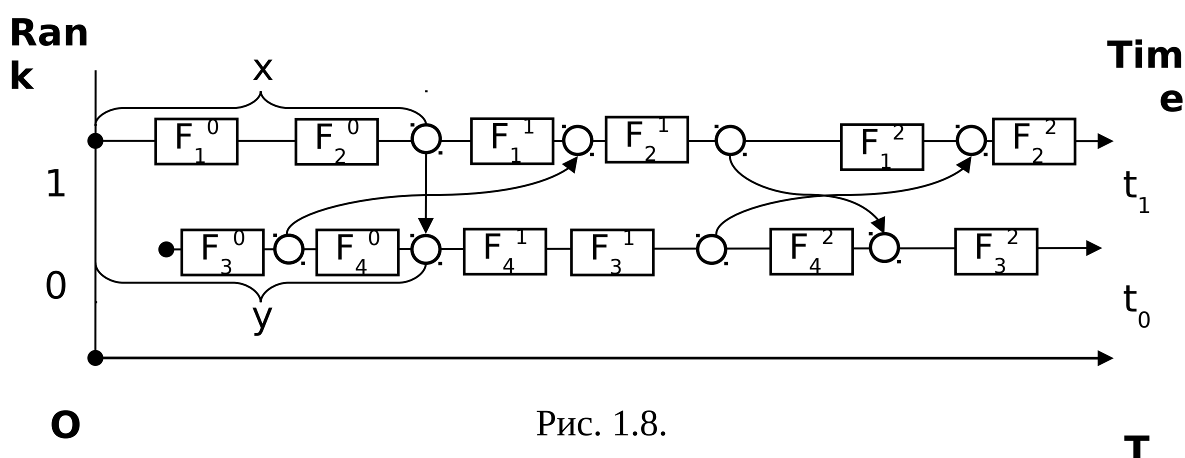


Рис. 1.8.

1.4. Описание алгоритма корректировки времени

Опишем предложенный алгоритм корректировки времени:

Пусть P – вектор, состоящий из трасс узлов мультикомпьютера, размерности size. Множество Away – события отправления ФД, которые на текущей стадии алгоритма находятся в пути от одного узла к другому. Вектор Block, размерности size - отвечает за блокировку просмотра каждой трассы узлов в отдельности. Вектор Drift, размерности size – сдвиги нулей исполнения, необходимые для устранения несоответствий.

```
Away = Ø;
Block [i] = false; i=1…size
Drift [i] = 0; i=1…size
    While (P !=0) {
        For ( i = 0; i < size; i++ ) {
            if (Block [i] = false) {
                p = P[i].first;
                If (p = sending message) {
                    Away = Away & p;
                    Delete P[i].first;
                }
                If (p = receiving massage)
                    If ( Away includes sending FD from p =:s ){
                        Away = Away/s;
                        Delete P[i].first;
                    }
                    Else    Block [i] = true;
            }
            Else {
                P[i].first = p;
                If ( Away includes sending FD from p =:s){
                    Drift[i] += s.time – p.time;
                    Away = Away/s;
                    Block [i] = false;
                }
            }
        }
    }
```

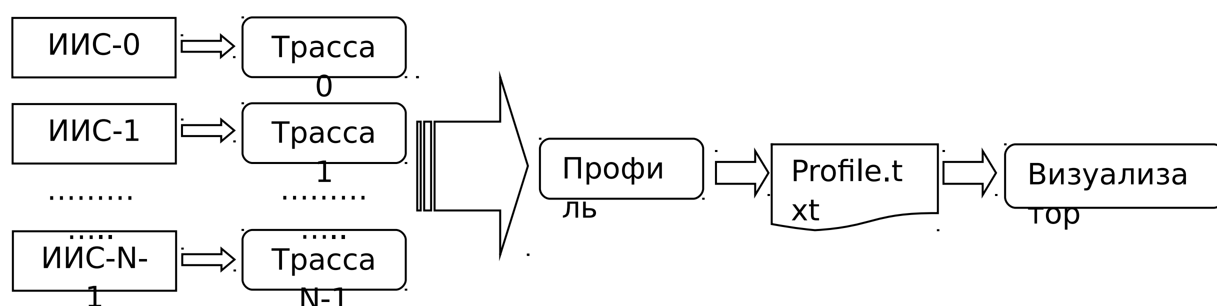
В результате, вектор Drift содержит величины, на которые достаточно сдвинуть нули исполнения узлов, чтобы устранить конфликты. Таким образом, мы получили способ перехода из локального времени узла к глобальному посредством сдвигов.

Итак, профиль формируется следующим образом: производятся нужные сдвиги для преобразования локальных времен к глобальному времени и события упорядочиваются по возрастанию.

Если часы на каких-то вычислительных узлах идут с разной скоростью, либо вследствие погрешностей при измерении измерение времени, то после корректировки времен в профиле могут остаться конфликты. Хотя, такая ситуация маловероятна, но по этой причине после корректировки профиля осуществляется проверка повторным просмотром профиля. Далее в работе ситуация с неточными часами не рассматривается.

2. РЕАЛИЗАЦИЯ

Предложенные в работе алгоритмы формирования и визуализации профиля были реализованы в виде программного комплекса «Профилировщик». Комплекс состоит из трех основных компонентов – конструкторов трасс, конструктора профиля и визуализатора. Схему работы ИС с профилировщиком можно проиллюстрировать рис. 2.1. Исполнительная система исполняет некоторую ФП на N узлах мультикомпьютера (ИИС-i). При этом конструктор трасс фиксирует события (см. раздел 1.2) и формирует из них трассы узла i. Затем трассы объединяются в профиль конструктором профиля. При этом конструктор профиля учитывает и корректирует возможную несинхронность часов на узлах мультикомпьютера. Сформированный профиль записывается в текстовый файл в заданном формате (описание см. в разделе 2.3.). Далее сохраненный профиль загружается в визуализатор, где наглядно представляется вниманию человека.



*ИИС-i – исполнение Фрагментированной Программы Исполнительной Системой на узле i.

Рис. 2.1.

С точки зрения пользователя использование ИС с профилировщиком выглядит следующим образом: пользователь запускает ФП в ИС, во время исполнения которой формируется профиль и записывается в текстовый файл. Далее пользователю необходимо лишь запустить визуализатор и загрузить туда собранный профиль и проанализировать отображенное исполнение.

Профилировщик был реализован на языке C++ с использованием интерфейса MPI для обмена сообщениями между узлами мультикомпьютера. Визуализатор реализован на платформе Qt.

2.1. Формат представления трасс и профиля

Напомним, что трасса – это последовательность событий, зафиксированных на одном узле мультикомпьютера. В системе событие трасс представлено в виде структуры `Event`:

```
struct Event{
  double time;
  vector<int> operation;
  ID id;
};
```

Где `time` – соответственно время свершения, `operation` – динамический массив параметров, описывающих событие, `id` – идентификатор фрагмента (может быть как и идентификатор ФВ, так и ФД).

Опишем подробнее вектор параметров описания события `operation`. Этот динамический массив устроен таким образом, что его элемент с заданным смещением имеет следующий смысл:

Элемент 0 – показывает номер узла, где произошло событие (0..N-1, где N – число узлов).

Элемент 1 – тип события, может принимать значения от 0 до 5, соответствующее следующему:

0 – удаление ФД;

1 – начало исполнения ФВ;

2 – конец исполнения ФВ;

3 – отправление ФД;

4 – получение ФД;

5 – создание ФД;

Элемент 2 (опциональный) – если тип события соответствует получению (отправки) ФД, то данный элемент хранит номер удаленного узла (отправителя или получателя), иначе этого элемента нет.

Поле `id` структуры `Event`– это уникальный идентификатор фрагмента, который используется в ИС. В случае типа события 1,2 – это идентификатор ФВ, иначе – ФД. В ИС данные о фрагменте представлены в виде строковой переменной имени фрагмента и набором целочисленных параметров (индексов). Для наших целей проще использовать один параметр фрагмента, поэтому все данные записываются в строковую переменную, которую и считаем идентификатором.

Рассмотрим формат представления профиля. События начала и завершения использования ФД ФВ требуют хранения дополнительного идентификатора ФВ, обрабатывающего ФД. Поэтому события профиля представлены в виде расширенной структуры `ExtendedEvent:`

```
struct ExtendedEvent{
    double time;
    vector<int> operation;
    ID id_1, id_2;
};
```

Отличия `ExtendedEvent` от `Event` заключается в том, что первый элемент `operation` может также принимать значения 6 и 7, соответствующие дополнительным событиям:

6 – начало использования ФД;

7 – конец использования/внесение изменений;

События типа 6 и 7 на этапе сбора трасс не фиксируются.

Итак, и трассы, и профиль являются динамическими массивами соответствующих элементов: трасса содержит элементы типа Event, а профиль – ExtendedEvent. Профиль также отличается тем, что он существует в единственном экземпляре и содержит события всех узлов мультикомпьютера.

2.2. Конструктор трасс

```
class Trace{
    private:
        vector<Event> events;
    public:
        void AddEvent(double time, int r1, int op, ID
df_id, int r2);
        void AddEvent(double time, int r, int op, ID
id);
        void AddEvent(Event ev, double drift);
        vector<Event>::iterator GetStart();
        Event GetEv( int it );
        size_t size();
        void DelEvent();
        size_t ActualSize();
        void* Pack();
        void WriteItDown(ofstream *fout, int N);
};
Profile Integrate( vector<Trace> RawSt);
Trace Unpack( void* buf);
```

В ключевых местах исходного кода ИС (то есть в местах, соответствующих определённым нами событиям) размещается код конструктора трасс, регистрирующий это событие. Он формирует новый объект типа `Event` и добавляет его в динамический массив `events` класса `Trace` с помощью методов `AddEvent`. На данном этапе времена событий локальны и берутся с показаний системных часов вычислительного узла.

После завершения исполнения программы, конструктор упаковывает/сериализирует свою трассу в непрерывный блок памяти при помощи метода `Pack` и отправляет его конструктору профиля, который

располагается на узле с номером 0. Сериализация трассы происходит следующим образом: сначала в буфер помещается целое число, равное количеству событий трассы, а далее по порядку записываются сами события следующим образом:

Сериализация события (в порядке записи в буфер): `time`, 3 элемента `operation`, идентификатор. Идентификатор представляет собой строковую переменную, размер которой заранее неизвестен и может быть разным для различных объектов. Поэтому в буфер сначала записывается его длина, а затем содержимое строки.

time	operaton[0]	operation[1]	operation[2]	id_size	id
double	int	int	int	int	ID

Заметим, что при сериализации событий считается, что элемент 2 массива `operation` есть у всех событий независимо от реальной длины массива.

Для передачи сериализованных данных используются функции MPI `MPI_Isend` и `MPI_Recv`.

2.3. Конструктор профиля

Конструктор профиля, находящийся на корневом узле, принимает и распаковывает трассы со всех узлов при помощи `Unpack` и размещает их в массиве трасс P. При десериализации, исходя из типа события `operation[1]` прочтенного ранее, элемент `operation[2]` либо обрабатывается, либо пропускается.

В соответствии с указанным в главе 2 алгоритмом корректировки времени, вычисляется вектор `Drift`, содержащий сдвиги преобразования локальных времен к глобальному времени. Вектор `Drift` представляется массивом `Drift` типа `double` размера N.

Формирование профиля происходит путем `Integrate` следующим образом:

1. конструктор просматривает первые из имеющихся событий каждой трассы и находит событие с минимальным глобальным временем.

2. Событие помещается в профиль и удаляется из трассы.
3. Если все события в профиле – конец, иначе – на шаг 1.

На этапе формирования профиля (точнее, на шаге 2) происходит добавление дополнительных событий типа 6 и 7. Если встречается событие начала ФВ, то при этом создаётся и добавляется в профиль новое событие начала использования ФД для всех ФД, которые используются этим ФВ. При этом созданным событиям устанавливается время срабатывания равное времени срабатывания события исполнения ФВ. Аналогично происходит и с событием конца исполнения ФВ, когда создаются соответственно события конца использования ФД. Входные и выходные ФД извлекаются из описания ФП по идентификатору.

Далее полученный профиль проходит повторную проверку на наличие конфликтов. В случае обнаружения таковых, пользователю сообщается о разном темпе хода часов в силу которого не удалось скорректировать времена.

В случае же отсутствия конфликтов, готовый профиль записывается в файл подобно тому, как упаковывались трассы узлов с той поправкой, что в файл информация записывается в виде текста: в первую строку записывается общее количество событий профиля, а затем построчно события – время, место, тип события, дополнительный параметр для событий отправки/приема (если есть), идентификатор фрагмента. Пример:

```
4
5 1 4 -1 df1
5 2 4 -1 df2
5 3 4 -1 df3
6 4 4 -1 df4
15 1 3 3 df1
22 3 3 1 df3
30 4 1 cf1
31 1 4 3 df3
35 1 1 cf2
```

Накладные расходы заведомо меньше, чем время исполнения программы, так как просмотров ФВ примерно столько же, но затрат на вычисление самих ФВ нет. Сложность формирования профиля линейная по количеству событий,

если не учитывать однократное упорядочение N элементов, где N – количество узлов мультикомпьютера.

2.4. Визуализатор

Визуализатор отображает для пользователя картину исполнения фрагментированной программы для облегчения ее анализа. Чтобы отобразить схему-график исполнения конкретной ФП требуется загрузить в визуализатор файл с профилем исполнения этой ФП.

Визуальное представление описано в главе 1. На рис. 2.2. представлен вид визуализатора с загруженным в него профилем. Визуализатор позволяет масштабировать визуальное представление ФП, что позволяет более детально увидеть ход исполнения программы. За масштабирование отвечает полоса прокрутки справа от отображения профиля.

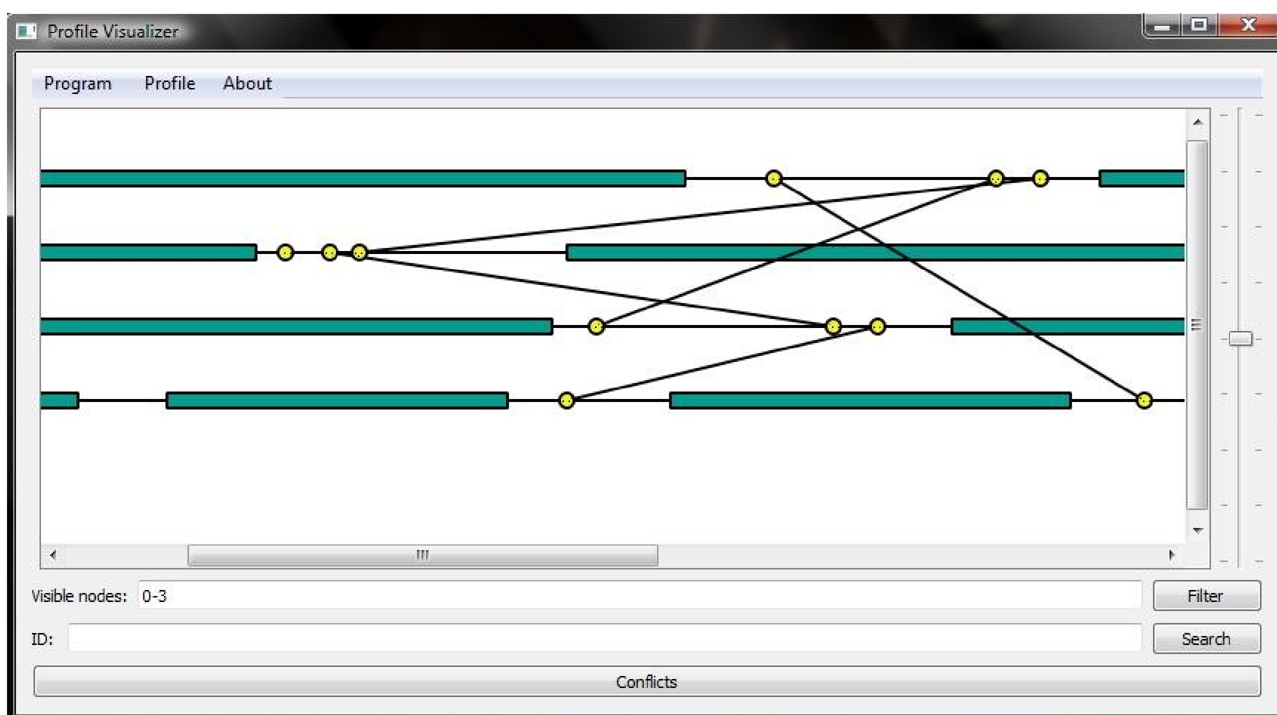


Рис. 2.2.

Далее представлены виды информации, которые можно получить из схемы-графика исполнения ФП путем указания/клика на соответствующий элемент схемы:

Элемент	Left Button	Right Button
Точка на временной оси	Информация о том, какие ФД хранились на	Нет

узла	выбранном узле в выбранный момент времени	
ФВ	Список входных и выходных ФД, время исполнения (рис. 2.3)	Список ФВ, обработавших ФД, попавшим выбранному ФВ на вход, а так же список тех ФВ, которые обработали выходные ФД выбранного ФВ
ФД или стрелка коммуникации	Идентификатор ФД	Путь ФД по узлам, то есть перемещения ФД и порядок изменения ФД ФВ.

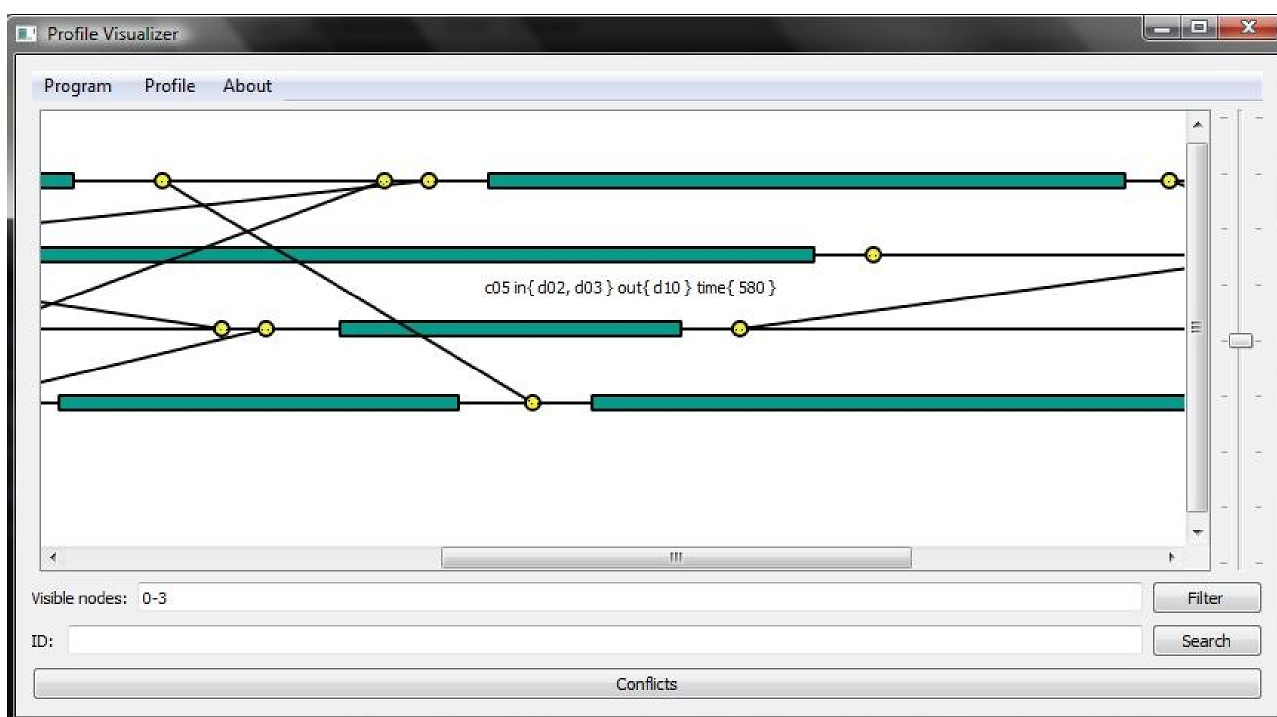


Рис. 2.3. Пример отображения данных о ФВ
при клике на его изображение на схеме.

Помимо отображения схемы-графика визуализатор имеет и другие возможности:

- Фильтр узлов (пример на рис. 2.4). Позволяет отображать только выбранные узлы. Выбор осуществляется записью номеров нужных узлов в текстовое поле – перечисление номеров узлов через запятую, а также интервалы вида “i-j”. Такой фильтр необходим, так как отображать множество узлов мультикомпьютера не актуально и с точки зрения пользователя – сложность

анализа, и с точки зрения визуализации – требуется время на построение. Коммуникации с узлами не подлежащих отображению показаны пунктирной линией: коммуникации отправления направлены вверх и вправо, а получения - вниз и влево.

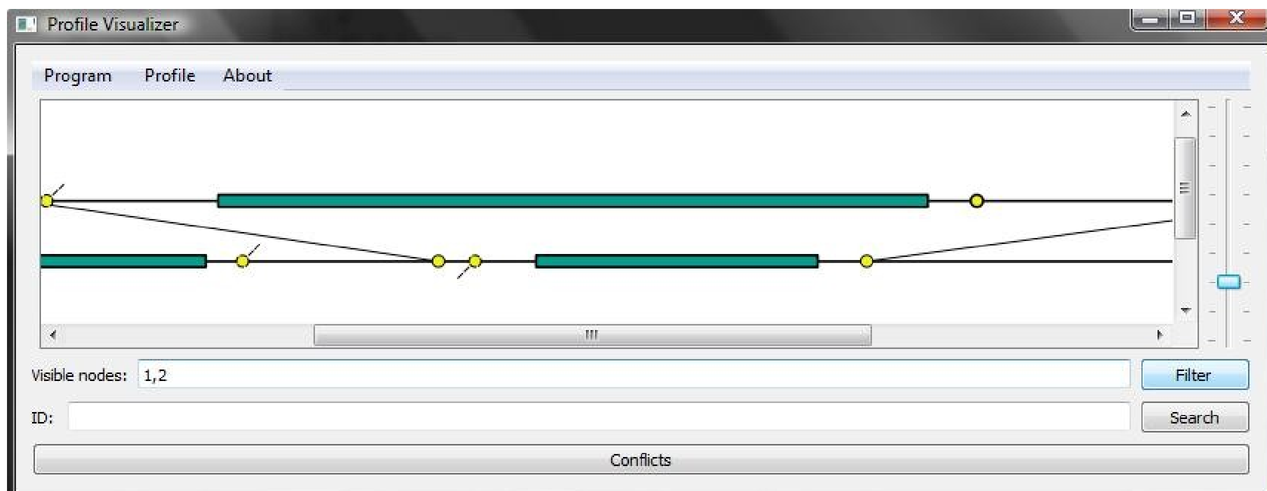


Рис. 2.4

- Поиск фрагментов по идентификатору – подсветка пути ФД при поиске ФД и исполнения ФВ при ФВ. На рис. 2.5. приведен пример выделения пути ФД при поиске по идентификатору d07.
- Автоматический поиск ошибок, описанных во введении как 1а – конфликтные ситуации. Так как отображению подлежат только события создания, отправки и удаления ФД, то ошибки такого рода описываются в виде текста в отдельном окне (при нажатии кнопки Conflicts). Например:

conflict: df0433

1.cf623 time 12987

2.cf085 time 13128

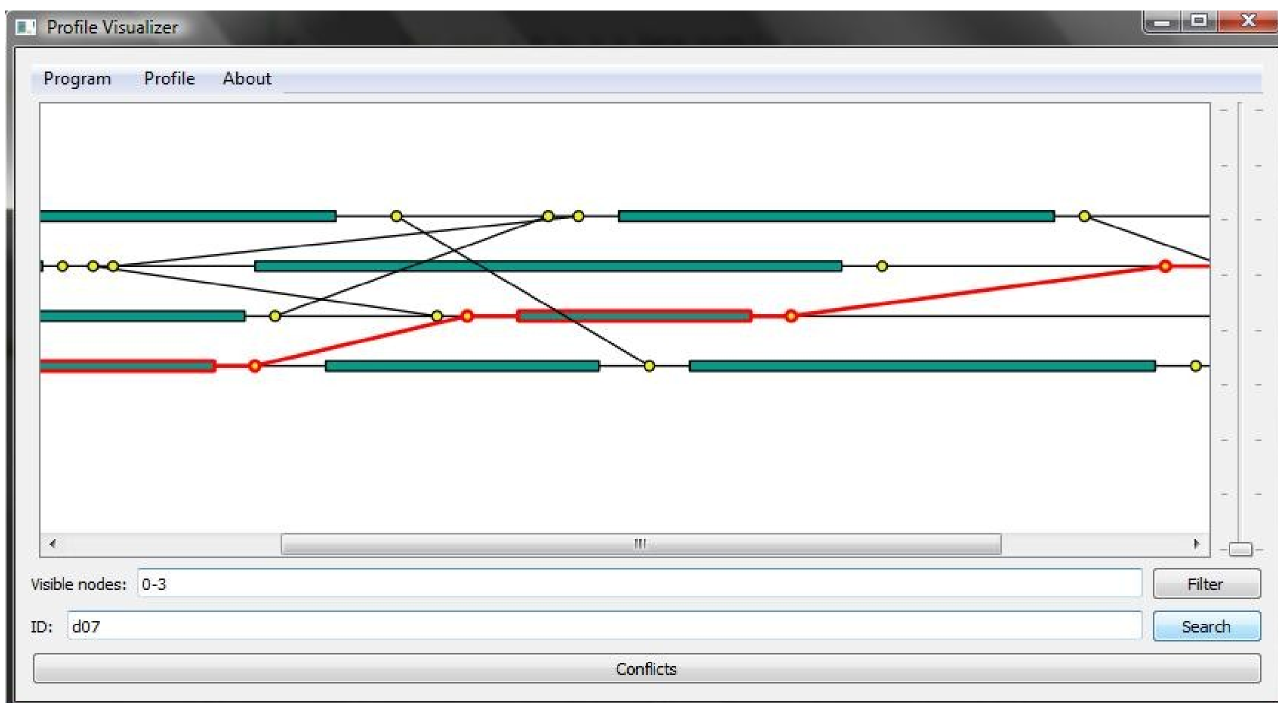


Рис. 2.5. Пример поиска ФД

Также можно подгрузить текст ФП для облегчения исправления ошибок. По умолчанию поле для текста программы не отображается, оно отделено от изображения профиля вертикальным разделителем.

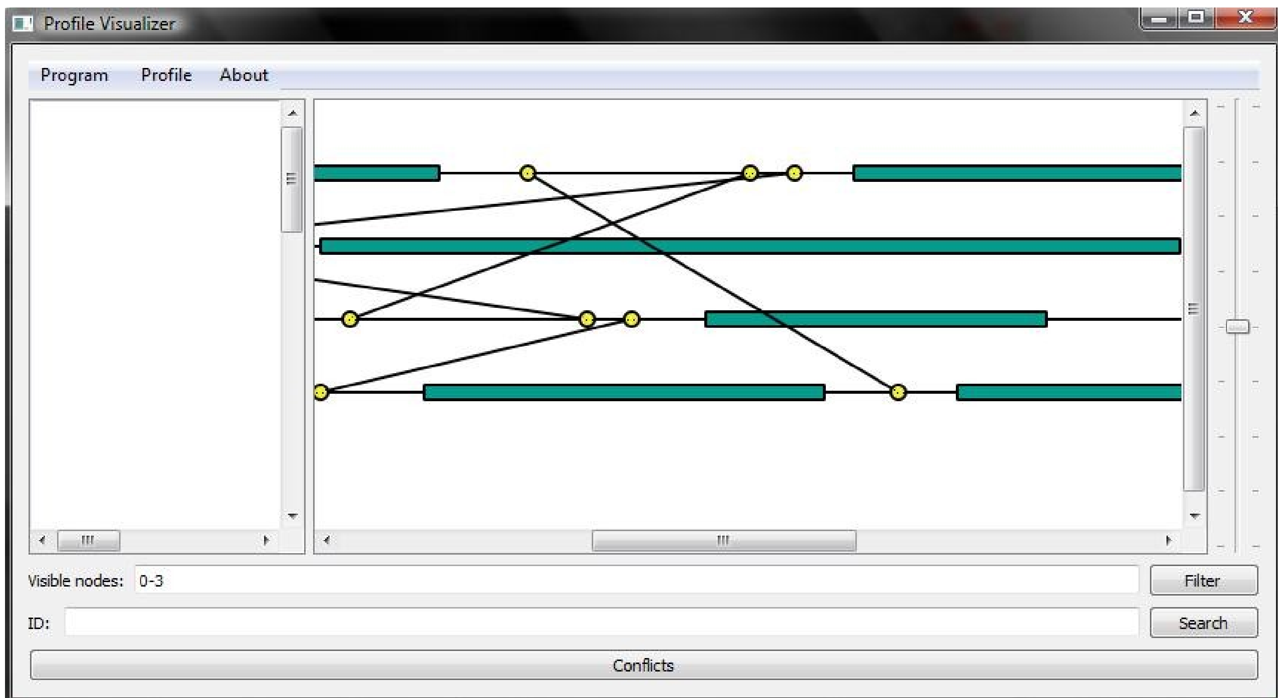


Рис. 2.6.

Особенности и ограничения реализации

При наличии профиля, содержащего большое число событий, требуются значительные затраты времени на обработку и построение графика. Для обеспечения нормального взаимодействия пользователя с интерфейсом в случае возможности действий во время графического построения целесообразно выделять отдельные потоки для обработки интерфейса и графики.

Также в реализации имеет место наличие излишней информации для пересылки – это номер узла в каждом событии трассы. Его может добавлять конструктор профиля.

ЗАКЛЮЧЕНИЕ

В работе проанализированы средства профилирования и визуализации исполнения параллельных программ, изучена ТФП. Предложен алгоритм контроля исполнения ФП, выявляющий потенциальные ошибки пользователя описанного вида. Предложен алгоритм корректировки профиля в случае несинхронности часов на различных узлах мультикомпьютера и доказана корректность его работы в случае, когда часы на различных узлах идут с одинаковой скоростью. Предложено визуальное представление исполнения ФП и показано его соответствие предъявляемым требованиям.

Реализован программный модуль для сбора и обработки информации, необходимой для анализа исполнения ФП. Реализован предложенный алгоритм контроля исполнения ФП. Реализован предложенный алгоритм коррекции профиля. Разработан и реализован визуализатор исполнения ФП в предложенном представлении.

На защиту выносится:

- алгоритмы профилирования исполнения ФП в виде модуля ИС LuNA;
- алгоритмы конструирования визуального представления профиля исполнения ФП, удовлетворяющего предъявляемым требованиям;
- алгоритм корректировки времени, устраняющий неточности связанные с несинхронностью часов;
- алгоритм контроля исполнения ФП;
- визуализатор профиля в предложенном представлении;

Дальнейшие планы:

Выполненная работа является первым шагом в решении проблемы отладки фрагментированных программ. Возможность анализировать наравне с профилем программы описание алгоритма предоставляет богатые возможности для автоматической отладки. В дальнейшем, работу планируется продолжить в магистратуре по следующим направлениям:

1. Расширение профилировщика функциональностью отладчика (например, фиксация значений заданных ФД).
2. Регулирование уровня детализации собираемой информации.
3. Разработка и реализация алгоритмов оптимизации исполнения ФП с учетом профиля.
4. Создание связи между ФП и визуализацией для упрощения изменения ФП.
5. Проверка, что реализованный при исполнении элемент поведения ФП определен в описании фрагментированного алгоритма.

ЛИТЕРАТУРА

1. Малышкин, В.Э Параллельное программирование мультикомпьютеров. – Изд-во НГТУ, Новосибирск, 2006. – 296 с.
2. Malyshkin V.E., Perepelkin V.A. Optimization of Parallel Execution of Numerical Programs in Luna Fragmented Programming System - Methods and Tools of Parallel Programming Multicomputers, LNCS 6083 - pp. 1-10, Springer, 2010
3. S.Kireev, V.Malyshkin: Fragmentation of Numerical Algorithms for Parallel Subroutines Library. The Journal of Supercomputing, Vol. 58, Issue 1, 2011 – Springer.
4. Malyshkin V., Perepelkin V. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System – Journal of Supercomputing, Springer (в печати)
5. Malyshkin V.E. Kraeva, M.A., Assembly technology for parallel realization of numerical models on mimd-multicomputers, the Int. Journal on Future Generation Computer Systems, Elsevier Science, NH 17 (2001), no. 6, 755-765.
6. Kalgin KV, Malyskin VE, Nechaev SP, Tschukin GA (2007) Runtime System for Parallel Execution of Fragmented Subroutines. 9th Int Conf Parallel Comput Technol, Springer Verlag, LNCS, Vol. 4671, pp. 544–552.
7. Malyshkin, V. E. Run-time system for parallel execution of fragmented subroutines. / V. E. Malyshkin [и др.] // Proceedings of the 9th International conference on Parallel Computing Technologies. – 2007. – Vol. 4671. – С. 544-552.
8. Кандидатская диссертация Романенко А.А. Средства отладки параллельных программ для мультикомпьютеров (алгоритмы реализации и разработка отладчика). 2004 г.
9. Automated Instrumentation and Monitoring System http://www.nas.nasa.gov/Software/AIMS/

10. Система визуализации Vampir. http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/software_werkzeuge_zur_unterstuetzung_von_programmierung_und_optimierung/vampir
11. Система генерации трасс Vampir Trace.http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/software_werkzeuge_zur_unterstuetzung_von_programmierung_und_optimierung/vampirtrace
12. Vampir - Perfomance Optimization http://www.vampir.eu/
13. Perfomance Visualisation for Parallel Programs. Viewers. http://www.mcs.anl.gov/perfvis/software/viewers/index.htm
14. The CXpert profiler. http://docs.hp.com/en/B6057-96002/ch05s03.html
15. Вальковский, В.А. Синтез параллельных программ и систем на вычислительных моделях. –Новосибирск: Наука. Сибирское отделение, 1988. – 128 с.
16. Tomas, G. Visualization of Scientific Parallel Programs / G. Tomas, C. W. Ueberhuber. – Berlin Heidelberg : Springer-Verlag, 1994. – 310 c.