**Аннотация**

Цель данной работы - разработка и реализация алгоритмов блока "Эксперт для мультипроцессора" в проекте "Экспериментальная система автоматизации распараллеливания". На основании результатов анализа последовательной программы и характеристик конкретного мультипроцессора "Эксперт" генерирует и оценивает различные варианты распределения вычислений и локализаций данных. После выбора наилучшего варианта он вставляет в последовательную программу директивы OpenMP, посредством которых она превращается в параллельную программу для мультипроцессора. Кроме того, эксперт выдает информацию о прогнозируемой эффективности распараллеливания.

1. Введение

1.1 Распараллеливание программ

1.2 Стандарт OpenMP

1.3 Распараллеливание для OpenMP

1.4 Cуть и актуальность проблемы

2. Цели дипломной работы

2.1 Цели проекта "Система автоматизации распараллеливания"

2.2 Цели "Эксперта для мультипроцессора"

2.3 Входные данные

2.4 Выходные и сохраняемые данные

3. Предыдущие решения "систем автоматизации распараллеливания"

4. Построение решения задачи

4.1 Формат входных данных

4.2 Основные аспекты работы эксперта

4.3 Оценочные функции

5. Пошаговая реализация "Эксперта"

5.1 Шаг 1. Предварительный анализ

5.2 Шаг 2. Выбор варианта распараллеливания

5.3 Шаг 3. Выбор варианта локализации

5.4 Шаг 4. Внесение конечных комментариев в Базу Данных и подсчет ускорения

5.5 Примеры работы алгоритма

5.5.1 Программа "Якоби"

5.5.2 Программа "Sor"

5.5.3 Программа "Модифицированный SOR"

5.5.4 Программа "Модифицированный Якоби"

6. Заключение

Перечень принятых терминов

Литература

# Введение

## Распараллеливание программ

Последние годы во всем мире происходит бурное внедрение вычислительных кластеров. Это вызвано тем, что кластеры стали общедоступными и дешевыми аппаратными платформами для высокопроизводительных вычислений. Одновременно резко возрос интерес к проблематике вычислительных сетей (GRID) и широко распространяется понимание того, что внедрение таких сетей будет иметь громадное влияние на развитие человеческого общества, сравнимое с влиянием на него появления в начале века единых электрических сетей. Поэтому, рассматривая проблемы освоения кластеров необходимо принимать во внимание и то, что они являются первой ступенькой в создании таких вычислительных сетей.

*Вычислительный кластер* – это *мультикомпьютер*, состоящий из множества отдельных компьютеров (*узлов*), связанных между собой единой коммуникационной системой. Каждый узел имеет свою локальную оперативную память. При этом общей физической оперативной памяти для узлов не существует. Если в качестве узлов используются *мультипроцессоры* (мультипроцессорные компьютеры с общей памятью), то такой кластер называется SMP-кластером. Коммуникационная система обычно позволяет узлам взаимодействовать между собой только посредством передачи сообщений, но некоторые системы могут обеспечивать и односторонние коммуникации - позволять любому узлу выполнять массовый обмен информацией между своей памятью и локальной памятью любого другого узла. Кластеры дешевле мультипроцессорных ЭВМ с распределенной памятью, в которых используются специальные коммуникационные системы и специализированные узлы.

Если все входящие в состав вычислительного кластера узлы имеют одну и ту же архитектуру и производительность, то мы имеем дело с *однородным* вычислительным кластером. Иначе – с *неоднородным*.

В настоящее время, когда говорят о кластерах, то часто подразумевают однородность. Однако, для того, чтобы сохранить высокий уровень соотношения производительность/стоимость приходится при наращивании кластера использовать наиболее подходящие в данный момент процессоры, которые могут отличаться не только по производительности, но и по архитектуре. Поэтому постепенно большинство кластеров могут стать неоднородными кластерами.

Неоднородность же вносит следующие серьезные проблемы. Различие в производительности процессоров требует соответствующего учета при распределении работы между процессами, выполняющимися на разных процессорах. Различие в архитектуре процессоров требует подготовки разных выполняемых файлов для разных узлов, а в случае различий в представлении данных может потребоваться и преобразование информации при передаче сообщений между узлами (не говоря уже о трудностях использования двоичных файлов).

Тем не менее, любой кластер можно рассматривать как единую аппаратно-программную систему, имеющую единую коммуникационную систему, единый центр управления и планирования загрузки. Вычислительные сети (GRID) объединяют ресурсы множества кластеров, многопроцессорных и однопроцессорных ЭВМ, принадлежащих разным организациям и подчиняющихся разным дисциплинам использования. Разработка параллельных программ для них усложняется из-за следующих проблем.

Конфигурация выделяемых ресурсов (количество узлов, их архитектура и производительность) определяется только в момент обработки заказа на выполнение вычислений. Поэтому программист не имеет возможностей для ручной настройки программы на эту конфигурацию. Желательно осуществлять настройку программы на выделенную конфигурацию ресурсов динамически, без перекомпиляции. К изначальной неоднородности коммуникационной среды добавляется изменчивость ее характеристик, вызываемая изменениями загрузки сети. Учет такой неоднородности коммуникаций является очень сложной задачей.

Все это требует гораздо более высокого уровня автоматизации разработки параллельных программ, чем тот, который доступен в настоящее время прикладным программистам.

С 1992 года, когда мультикомпьютеры стали самыми производительными вычислительными системами, резко возрос интерес к проблеме разработки для них параллельных прикладных программ. К этому моменту уже было ясно, что трудоемкость разработки прикладных программ для многопроцессорных систем с распределенной памятью является главным препятствием для их широкого внедрения. За прошедший с тех пор период предложено много различных подходов к разработке параллельных программ, созданы десятки различных языков параллельного программирования и множество различных инструментальных средств.

На Рис. 1 описаны основные особенности симметричных мультипроцессорных систем (SMP). Для данных систем, как модель программирования для мультипроцессоров, возникла модель параллелизма по управлению (в западной литературе используется и другое название – модель разделения работы, work-sharing model). На мультипроцессорах в качестве модели выполнения используется модель общей памяти. В этой модели параллельная программа представляет собой систему нитей, взаимодействующих посредством общих переменных и примитивов синхронизации. Нить (по-английски "thread") – это легковесный процесс, имеющий с другими нитями общие ресурсы, включая общую оперативную память.

|  |  |
| --- | --- |
| Архитектура | Система состоит из нескольких однородных процессоров и массива общей памяти (обычно из нескольких независимых блоков). Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины (базовые 2-4 процессорные SMP-сервера), либо с помощью crossbar-коммутатора (HP 9000). Аппаратно поддерживается когерентность кэшей. |
| Примеры | HP 9000 V-class, N-class; SMP-cервера и рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu и др.). |
| Масштабируемость | Наличие общей памяти сильно упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число - не более 32 в реальных системах. Для построения масштабируемых систем на базе SMP используются кластерные или NUMA-архитектуры. |
| Операционная система | Вся система работает под управлением единой ОС (обычно UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы/нити по процессорам (scheduling), но иногда возможна и явная привязка. |
| Модель программирования | Программирование в модели **общей памяти**. (POSIX threads, OpenMP). Для SMP-систем существуют сравнительно эффективные средства автоматического распараллеливания. |

Рис. 1. Симметричные мультипроцессорные системы.

Основная идея модели параллелизма по управлению заключалась в следующем. Вместо программирования в терминах нитей предлагалось расширить языки специальными управляющими конструкциями – параллельными циклами и параллельными секциями. Создание и уничтожение нитей, распределение между ними витков параллельных циклов или параллельных секций (например, вызовов процедур) – все это брал на себя компилятор.

Первая попытка стандартизовать такую модель привела к появлению в 1990 году проекта языка PCF Fortran (проект стандарта X3H5). Однако, этот проект тогда не привлек широкого внимания и, фактически, остался только на бумаге. Возможно, что причиной этого было снижение интереса к мультипроцессорам и всеобщее увлечение мультикомпьютерами и HPF.

Однако, спустя несколько лет ситуация сильно изменилась. Во-первых, успехи в развитии элементной базы сделали очень перспективным и экономически выгодным создавать мультипроцессоры. Во-вторых, широкое развитие получили мультикомпьютеры с DSM (distributed shared memory - распределенная общая память), позволяющие программам на разных узлах взаимодействовать через общие переменные также, как и на мультипроцессорах (Convex Exemplar, HP 9000 V-class, SGI Origin 2000). В-третьих, не оправдались надежды на то, что HPF станет фактическим стандартом для разработки вычислительных программ.

Крупнейшие производители компьютеров и программного обеспечения объединили свои усилия и в октябре 1997 года выпустили описание языка OpenMP Fortran – расширение языка Фортран 77. Позже вышли аналогичные расширения языков Си и Фортран 90/95.

Параллельно развивалась модель передачи сообщений. Обобщение и стандартизация различных библиотек передачи сообщений привели в 1993 году к разработке стандарта MPI (Message Passing Interface). В модели передачи сообщений параллельная программа представляет собой множество процессов, каждый из которых имеет собственное локальное адресное пространство. Взаимодействие процессов - обмен данными и синхронизация - осуществляется посредством передачи сообщений. Однако разработчики MPI подвергаются и суровой критике за то, что интерфейс получился слишком громоздким и сложным для прикладного программиста.

Успешное внедрение OpenMP на мультипроцессорах и DSM-мультикомпьютерах резко активизировало исследования, направленные на поиски путей распространения OpenMP на мультикомпьютеры, кластеры и сети ЭВМ. Эти исследования сосредоточились, в основном, на двух направлениях:

* расширение языка средствами описания распределения данных;
* программная реализация системы DSM, использующей дополнительные указания компилятора, вставляемые им в выполняемую программу.

Первое направление представляется гораздо более перспективным для кластеров и сетей ЭВМ, однако трудно ожидать появления в ближайшие годы время стандарта нового языка (расширенного OpenMP). Поэтому все шире начинает использоваться гибридный подход, когда программа представляет собой систему взаимодействующих MPI-процессов, а каждый процесс программируется на OpenMP.

Такой подход имеет преимущества с точки зрения упрощения программирования в том случае, когда в программе есть два уровня параллелизма – параллелизм между подзадачами и параллелизм внутри подзадачи. Такая ситуация возникает, например, при использовании многообластных (многоблочных) методов решения вычислительных задач. Программировать на MPI сами подзадачи гораздо сложнее, чем их взаимодействие, поскольку распараллеливание подзадачи связано с распределением элементов массивов и витков циклов между процессами. Организация же взаимодействия подзадач таких сложностей не вызывает, поскольку сводится к обмену между ними граничными значениями. Нечто подобное программисты делали раньше на однопроцессорных ЭВМ, когда для экономии памяти на каждом временном шаге выполняли подзадачи последовательно друг за другом.

Широкое распространение SMP-кластеров также подталкивает к использованию гибридного подхода, поскольку использование OpenMP на мультипроцессоре может для некоторых задач (например, вычислений на неструктурных сетках) дать заметный выигрыш в эффективности.

Основной недостаток этого подхода также очевиден - программисту надо знать и уметь использовать две разные модели параллелизма и разные инструментальные средства [1].

## Стандарт OpenMP

Интерфейс OpenMP [4] задуман как стандарт для программирования на масштабируемых SMP-системах в модели общей памяти (shared memory model). В стандарт OpenMP входят спецификации набора директив компилятора, процедур и переменных среды. Разработкой стандарта занимается организация OpenMP ARB (ARchitecture Board), в которую вошли представители крупнейших компаний - разработчиков SMP-архитектур и программного обеспечения. Спецификации для языков Fortran и C/C++ появились соответственно в октябре 1997 года и октябре 1998 года.

program compute\_pi

parameter (n = 1000)

integer i

double precision w,x,sum,pi,f,a

f(a) = 4.d0/(1.d0+a\*a)

w = 1.0d0/n

sum = 0.0d0;

**!$OMP PARALLEL DO PRIVATE(x), SHARED(w)**

**!$OMP& REDUCTION(+:sum)**

do i=1,n

x = w\*(i-0.5d0)

sum = sum + f(x)

enddo

pi = w\*sum

print \*,'pi = ',pi

stop

end

Рис. 2. OpenMP программа вычисления числа Пи.

В OpenMP реализована "золотая середина" между двумя крайностями - полностью автоматическим распараллеливанием и кодированием с явным вызовом нитей; это подход, который основывается на распараллеливании обычных последовательных программ компилятором, но "с подсказками" пользователя, помещающего в исходный текст программы директивы в виде комментариев. Эти директивы указывают компилятору на параллелизм на уровне циклов и отдельных фрагментов программы. За счет этого OpenMP идеально подходит для разработчиков, желающих быстро распараллелить свои вычислительные программы с большими параллельными циклами.

Разработчик не создает новую параллельную программу, а просто последовательно добавляет в текст последовательной программы OpenMP-директивы. На Рис. 2 показано удобство OpenMP: необходимо вставить всего 2 строчки в программу, вычисляющую число пи, чтобы она стала параллельна.

Автоматическое распараллеливание весьма затруднено тем обстоятельством, что компилятору трудно (если вообще возможно) распознать взаимозависимости по данным, возможности возникновения тупиков, ситуаций типа "гонки" (race condition) и другие подобные ситуации, которые зачастую определяются поведением программы уже на этапе выполнения. Поэтому возможность получить "подсказку" от пользователя может быть очень ценна.

Однако стандарт OpenMP не ограничивается только набором директив, а специфицирует API [10], который, кроме директив компилятору, включает набор функций системы поддержки выполнения OpenMP-программ, а также переменные окружения. Далее мы рассмотрим основные особенности OpenMP применительно к языку Fortran [9]. Существуют аналогичные средства для языков Си/C++. В рамках OpenMP стандартная последовательная модель языка Fortran расширяется параллельными конструкциями SPMD (Single Program, Multiple Data), которые наиболее близки к традиционной "последовательной" идеологии.

В OpenMP используется модель параллельного выполнения fork/join. Программа начинает выполняться как один процесс, который в ОpenMP называется главной нитью (master thread). Этот процесс выполняется последовательно до тех пор, пока он не дойдет до первой параллельной конструкции (в простейшем случае - области, заключенной между директивами PARALLEL и END PARALLEL). В этот момент создается "бригада" (team) нитей, а "бригадиром" для нее является главная нить. Внутри бригады главная нить имеет номер 0, а число нитей в бригаде задается переменной окружения или может изменяться динамически перед началом выполнения параллельной области путем вызова соответствующей функции системы поддержки выполнения OpenMP-программ. Однако внутри параллельной области число нитей фиксировано.



Последовательная секция

Параллельная секция

Последовательная секция

Рис. 3. Процесс исполнения OpenMP программы.

**!$OMP PARALLEL**

**!$OMP END PARALLEL**

Кусок программы, заключенный между директивами PARALLEL и END PARALLEL, выполняется параллельно всеми нитями бригады (см. Рис. 3). Операторы программы, логически заключенные между этой парой директив, образуют, в терминологии OpenMP, лексическую (статическую) область действия соответствующей параллельной конструкции. Поскольку указанный блок программы может содержать вызовы подпрограмм, которые тоже могут выполняться параллельно, вводится понятие динамической области действия, который включает и подпрограммы, вызываемые из данного блока [3].

После завершения выполнения параллельной конструкции нити бригады синхронизируются, а выполнение продолжает только главная нить. Естественно, в программе может быть много параллельных конструкций; соответственно бригады нитей могут образовываться не один раз. OpenMP предусматривает возможность вложения параллельных конструкций (пар PARALLEL/END PARALLEL). Многие "ключи" (clauses) в директивах OpenMP позволяют указать атрибуты данных, действующие в период выполнения соответствующей параллельной конструкции.

По умолчанию, в параллельном регионе частными (приватными) являются индексы параллельных циклов DO, все остальные переменные, по умолчанию, общие. Однако это правило умолчания можно изменить, указав ключ DEFAULT(PRIVATE) или вовсе отменить, если задать DEFAULT(NONE). Счетчики фортрановских DO-циклов следует делать приватными (PRIVATE) для каждой нити бригады. Такие приватные объекты (простые переменные, массивы и т.д.) с точки зрения их обработки эквивалентны тому, как если бы для каждой нити имелась бы декларация нового объекта того же типа. Все ссылки на исходный объект в лексической области действия параллельной конструкции заменяются на ссылки на этот новый приватный объект.

Переменные, определенные как PRIVATE, при входе в параллельную конструкцию являются неопределенными для любой нити бригады. При использовании описателя FIRSTPRIVATE приватные копии переменных инициализируются данными из исходного объекта, существующего до входа в параллельную конструкцию. LASTPRIVATE отличается от параметра PRIVATE тем, что после выхода из цикла или из последней секции конечные значения, перечисленные в списке переменных, будут доступны для использования.

Директива THREADPRIVATE(список переменных) объявляет приватными в нитях параллельного региона переменные, описанные в программе. Директива указывается сразу после описания соответствующих переменных и не влияет на работу с ними вне параллельного региона. С ней связана директива COPYIN(список переменных)- для каждой нити параллельного региона создаются копии переменных, описанных в директиве threadprivate, копиям присваиваются текущие значения оригиналов. Имена, указанные в списках директивы threadprivate и параметра copyin, должны совпадать. COPING применяется с директивой parallel.

Существует возможность использования условного оператора: *IF(cкалярное\_логическое\_выражение)*.

Если этот ключ указан, то соответствующая параллельная область кодов будет выполняться несколькими нитями только при условии, что "скалярное\_логическое\_выражение" истинно. Например, указав IF(N.GT.1000), можно заставить компилятор породить такую программу, что во время выполнения проверяется размерность (N) и программа выполнялась бы параллельно, только если эта размерность достаточно велика - больше 1000. Это позволяет избегать распараллеливания при маленьких размерностях, когда оно может оказаться невыгодным из-за большой доли накладных расходов.

Важным компонентом стандарта OpenMP является набор подпрограмм времени выполнения и переменных окружения, задающих среду OpenMP:

* subroutine OMP\_SET\_NUM\_THREADS(N) - устанавливает число нитей, равное N.
* integer function OMP\_GET\_NUM\_THREADS() - возвращает число нитей в бригаде.
* integer function OMP\_GET\_THREAD\_NUM() -возвращает номер "текущей" нити (из которой произошел вызов функции) в бригаде.

Кроме указаний на распределение работ между нитями, OpenMP имеет директиву SINGLE/END SINGLE, которая разрешает выполнять заключенный в нее блок Fortran-программы только одной нити.

!$OMP DO [ключ[[,]ключ...]

DO-цикл языка Fortran

!$OMP END DO [NOWAIT]

Рис. 4. Директива DO.

Для распараллеливания циклов существует директива DO (см. Рис. 4). В качестве "ключей" можно указать описатели PRIVATE(список), FIRSTPRIVATE(список), SHARED(список), ключ REDUCTION, а также ключи ORDERED и SCHEDULE.

Ключ SCHEDULE используется для указания дисциплины планирования выполнения цикла нитями и имеет вид: *SCHEDULE(тип[,M])*

Он указывает, каким образом итерации цикла будут распределены для выполнения между нитями бригады. В качестве параметра "тип" можно указать следующие:

* STATIC [,m] - блочно-циклическое распределение итераций: первый блок из m итераций выполняет первая нить, второй блок - вторая и т.д. до последней нити, затем распределение снова начинается с первой нити; по умолчанию значение m равно 1;
* DYNAMIC [,m] - динамическое распределение итераций с фиксированным размером блока: сначала все нити получают порции из m итераций, а затем каждая нить, заканчивающая свою работу, получает следующую порцию опять-таки из m итераций;

!$OMP PARALLEL DO REDUCTION(+: sum)

!$OMP& PRIVATE(x)

do i=1,n

x=a(i)

sum=sum+f(x)

enddo

!$OMP END PARALLEL DO

Рис. 5. Пример использования REDUCTION.

* GUIDED [,m] - динамическое распределение итераций блоками уменьшающегося размера; аналогично распределению DYNAMIC, но размер выделяемых блоков все время уменьшается, что в ряде случаев позволяет аккуратнее сбалансировать загрузку нитей;
* RUNTIME - способ распределения итераций цикла выбирается во время работы программы в зависимости от значения переменной OMP\_SCHEDULE.

!$OMP SECTIONS [ключ[[,]ключ...]

!$OMP SECTION

блок Fortran-операторов

[!$OMP SECTION

блок Fortran-операторов]

...

!$OMP END SECTIONS [NOWAIT]

Рис. 6. Структура директивы SECTIONS.

Если в OpenMP-директиве END DO задано NOWAIT, в конце цикла нити не синхронизируются. Ключ REDUCTION директивы !$OMP PARALLEL DO обозначает редукционную переменную (см. Рис. 5).

В OpenMP имеется также возможность использовать сокращения, позволяющие объединять инструкции PARALLEL/END PARALLEL и DO/ENDDO в пару PARALLEL DO/END PARALLEL DO.

Директива SECTIONS (см. Рис. 6) используется для распределения работы между нитями в "неитерационном" случае. Ключи, как и ранее, могут указывать на тип переменных (PRIVATE, FIRSTPRIVATE и т.д.), а также содержать ключ REDUCTION. OpenMP-конструкция SECTIONS позволяет нескольким нитям выполнять параллельно обычный линейный участок программы на Фортране, разбитый на "секции" - блоки операторов языка. Эти секции могут содержать в себе, в частности, вызовы фортрановских подпрограмм.

!$OMP CRITICAL [ (<имя\_критической\_секции>) ]

...

!$OMP END CRITICAL [ (< имя\_ критической\_секции >) ]

Рис. 7. Директива CRITICAL.

Богатые возможности предоставляют средства OpenMP и для синхронизации. Соответствующие директивы позволяют, в частности, установить барьер (!$OMP BARRIER) или обеспечить синхронизацию (!$OMP ATOMIC), предохраняющую от одновременной записи несколькими нитями в одно и то же поле оперативной памяти. Это позволяет предотвратить ситуацию гонки. Другая полезная директива, о которой стоит упомянуть - это !$OMP FLUSH. Она заставляет в соответствующей точке программы получить "согласованное" между нитями состояние оперативной памяти (при этом переменные из регистров будут записаны в память, сбросятся буферы записи и т.д.). Пара директив MASTER: END MASTER выделяет участок кода, который будет выполнен только нитью-мастером. Остальные нити пропускают данный участок и продолжают работу с выполнения оператора, расположенного следом за директивой END MASTER. С помощью директивы CRITICAL оформляется критическая секция программы. В каждый момент времени в критической секции может находиться не более одной нити (см. Рис. 7). Если критическая секция уже выполняется какой-либо нитью P0, то все другие нити, выполнившие директиву для секции с данным именем, будут заблокированы, пока нить P0 не закончит выполнение данной критической секции. Как только P0 выполнит директиву END CRITICAL, одна из заблокированных на входе нитей войдет в секцию. Если на входе в критическую секцию стояло несколько нитей, то случайным образом выбирается одна из них, а остальные заблокированные нити продолжают ожидание. Все неименованные критические секции условно ассоциируются с одним и тем же именем.

Директива ORDERED заставляет коды Fortran, заключенные внутри пары ORDERED/END ORDERED, выполняться в "естественном" порядке (в той, в которой итерации цикла выполняются при нормальном последовательном выполнении). Эта пара директив может присутствовать только в динамической области действия директивы DO (или PARALLEL DO), в которой указан ключ ORDERED. Такие директивы могут использоваться для организации печати в нормальной - скажем, по возрастанию счетчика цикла- последовательности [2].

## Распараллеливание для OpenMP

Преобразование последовательной программы в параллельную – процесс нетривиальный, неоднозначный и неформальный. Распараллеливание в OpenMP- программу, в конечном счете, сводится к принятию решения по распределению нитей и вычислений и расстановке в текст программы соответствующих OpenMP-директив. Разные способы объявления данных, распределения вычислений и синхронизации нитей могут иметь разную эффективность, причем эффективность конкретного распределения может зависеть как от конфигурации системы, так и от размера задачи.

Из этого следует, что:

* трудно рассчитывать на создание системы полностью автоматизирующей это преобразование, т.е. распараллеливание представляется как итерационный процесс взаимодействия пользователя с системой;
* система не должна ограничивать выбор, а по возможности, должна и подсказывать варианты распараллеливания;
* система должна допускать вмешательство пользователя, т.е. принятие решений самим пользователем;

## Cуть и актуальность проблемы

Преимущества OpenMP:

1. Разработчик не создает новую параллельную программу, а просто последовательно добавляет в текст последовательной программы OpenMP-директивы.
2. OpenMP - достаточно гибкий механизм, предоставляющий разработчику большие возможности контроля над поведением параллельного приложения.
3. Предполагается, что OpenMP-программа на однопроцессорной платформе может быть использована в качестве последовательной программы, т.е. нет необходимости поддерживать последовательную и параллельную версии. Директивы OpenMP просто игнорируются последовательным компилятором.
4. Одним из достоинств OpenMP его разработчики считают поддержку так называемых "orphan" (оторванных) директив, то есть директивы синхронизации и распределения работы могут не входить непосредственно в лексический контекст параллельной области.
5. OpenMP - стандарт, который разработан представительным форумом (HP, IBM, Intel, SGI, Sun, Fujitsu, NEC) для программирования мультипроцессоров и DSM-систем
6. Поддерживается в компиляторах языков Fotran, C, C++ для всех основных типов процессоров.

Однако, "легкость" программирования на OpenMP возможно еще упростить, за счет "системы автоматизации распараллеливания". Если "последовательный" программист будет видеть анализ своей программы и рекомендации по распараллеливанию, "умственные затраты" на вставку распараллеливающих директив сведутся к минимуму или пропадут вообще. Другой вариант использования системы - анализ эффективности уже распараллеленной программы или сравнение ее с другими вариантами распараллеливания. В этом случае заметно облегчается поиск оптимального распараллеливания. Таким образом, при существовании такой системы любой последовательный программист сможет без дополнительной подготовки преобразовывать свой код к "параллельному" варианту. Это особенно актуально для программистов, производящих сложные вычисления, занимающие огромное количество времени. Использование мультипроцессорных систем должно дать заметное ускорение таких вычислений.

# Цели дипломной работы

## 

## Цели проекта "Система автоматизации распараллеливания"

Данная дипломная работа является лишь частью проекта автоматизации распараллеливания программ, поэтому сначала остановимся на проекте.

DVM рекомендации

OpenMP рекоммендации

Последовательная программа

Пользователь

**program compute\_pi**

**parameter (n = 1000)**

**integer i**

**double precision w,x,sum,pi,f,a**

**f(a) = 4.d0/(1.d0+a\*a)**

**w = 1.0d0/n**

**…**

**end**



**program compute\_pi**

**parameter (n = 1000)**

**integer i**

**double precision**

**!$OMP PARALLEL DO PRIVATE(x), HARED(w)**

**w,x,sum,pi,f,a**

**f(a) = 4.d0/(1.d0+a\*a)**

**w = 1.0d0/n**

**…**

**end**

База данных

**program compute\_pi**

**parameter (n = 1000)**

**integer i**

**double precision**

**DVM PARALLEL**

**w,x,sum,pi,f,a**

**f(a) = 4.d0/(1.d0+a\*a)**

**w = 1.0d0/n**

**…**

**end**

Рис. 8. Проект: Система автоматизации распараллеливания.

На Рис. 8 схематически показаны составляющие проекта: "Анализ программы", "Эксперт для мультипроцессора" (OpenMP) и "Эксперт для кластера" (DVM).

Пользователь создает последовательную программу на языке Fortran. В итоге работы системы, пользователь получает информацию для анализа:

* дерево циклов; для каждого цикла – его параметры и атрибуты (результаты анализа);
* описания данных;
* описание использования массивов в циклах и найденные зависимости;
* список предложенных (в том числе рассмотренных, но отвергнутых) вариантов распределения (задача "Эксперта");
* созданные и модифицируемые пользователем наборы конфигураций целевой машины и размеров задач;
* для каждого предложенного варианта распараллеливания (всех конфигураций и размеров) и каждого цикла и всей программы, как корня дерева циклов, – оценки времени выполнения и качества распараллеливания для данного варианта (задача "Эксперта");

Программа, поступая в систему автоматизации распараллеливания, проходит анализ, на основании которого формируется База Данных, в которую входят: дерево циклов; описания массивов, описание использования массивов в циклах; дополнительные комментарии.

С информацией из базы данных "Эксперты" ищут самые оптимальные способы распараллеливания и добавляют информацию о них в базу данных. Код программы на выходе системы не изменяется, а всего лишь добавляются комментарии, где рекомендуется вставить директивы OpenMP или DVM (возможно несколько лучших вариантов), насколько будут эффективны эти вставки, а также информация из анализа.

## Цели "Эксперта для мультипроцессора"

Цель данного диплома – разработка программы, реализующей "Эксперт для мультипроцессора" в проекте. На основании данных, полученных на стадии анализа, и характеристик конкретного мультипроцессора "Эксперт" должен занести в базу данных информацию о комментариях (директивах OpenMP), которые должны быть вставлены в программу, которая получится на выходе системы, а также информацию о прогнозируемой эффективности распараллеливания. Таким образом, "Эксперт" генерирует, оценивает и выбирает лучшие варианты распределения вычислений и локализации данных. Эксперт не изменяет код программы, а лишь вносит в базу данных информацию о "наилучших рекомендациях по распараллеливанию". Эта информация впоследствии и будет выдана пользователю.

## Входные данные

**Описание цикла** содержит:

* указание на переменную цикла;
* первое, последнее значение;
* наличие в цикле операторов ввода-вывода, побочных выходов и т.п.
* оценка трудоемкости одного витка цикла;
* признак тесной вложенности;
* список обращений к массивам и переменным.

**Дерево циклов** – множество циклов с отношением "*непосредственно вложен в*". Корень "дерева циклов", уровень программы, сам циклом не является, но формально может считаться "циклом" без переменной цикла с однократным выполнением.

**Список массивов (и простых переменных).**

Для каждого массива:

* идентификатор;
* размер по каждому измерению;
* список обращений к массиву.

**Вариант задачи:**

* размерности массивов по каждому измерению (уже как константы);
* число итераций;
* или значения внешних переменных, от которых зависят эти величины.

**Вариант массива процессоров:**

* число измерений и размер по каждому измерению (решетка процессоров).

**Обращения к массивам (и переменным) в цикле** – отношение "к <*массиву*> обращаются в <*цикле*>". Для каждой пары <массив, цикл>:

* нет (не создает) зависимости ни между итерациями, ни внутри;
* зависимость внутри одной итерации;
* REDUCTION (зависимость между итерациями создают операторы специального вида, типа *a=a+xi*);
* Private, Lastprivate, Firstprivate – приватные переменные
* возникающая из-за него зависимость между итерациями общего вида*.*

**Список всех обращений**, т.е. наборов индексных выражений, желательно без повторений. Про каждое надо указать:

* чтение или/и присваивание;
* индексные выражения;

## Выходные и сохраняемые данные

Выходные данные представляют собой:

* Вариантраспараллеливания программы. Вариант представляет собой набор правил получения OpenMP-программы. Оценка эффективности выполняется для данного варианта распараллеливания и данного варианта запуска.
* Оценка данного варианта распараллеливания и данного варианта запуска: параметры производительности цикла, программы в целом и эффективности распараллеливания.
* Информация, объясняющая, почему выбрано данное распределение. Для каждой вставленной директивы производится вычисление ускорения, полученного за счет этой директивы.

# 

# Предыдущие решения "систем автоматизации распараллеливания"

Здесь будет произведен краткий обзор систем, в которых разработчики и рабочие группы достигли значительных продвижений в решении задачи.

1. **Interactive Parallelizing Assistance Tool for OpenMP** [7]

Aachen University, Германия. 2003 г.

Система анализирует эффективность OpenMP программы и выдает результаты анализа пользователю. Система работает только с циклами. Для конкретного цикла можно посмотреть, какие директивы OpenMP возможны для вставки. По результатам анализа пользователь сам вставляет директивы или меняет код программы. Есть опция выбора отдельного участка программы для проверки на возможность распараллеливания.

1. **Interprocedural Parallelizing Compiler WPP and Analysis Information Visualization tool Aivi** [8]

Hitachi, Ltd. Япония. 2000 г.

Система производит межпроцедурный анализ: анализ скаляров и массивов, клонирование процедур, анализ вызовов функций. Система работает только с циклами. Генерация OpenMP директив происходит автоматически. В систему входит мощный инструмент визуализации, показывающий пользователю результаты анализа.

1. **The ParaWise Expert Assistant** [6]

University of Greenwich, Великобритания. Август 2004.

Представляет собой один из инструментов системы ParaWise/CAPO [5]. На основании инструментов системы, проводящих анализ программы, "Эксперт- ассистент" подсказывает: возможно ли распараллеливание, и представляет некоторые варианты распараллеливания (вставки OpenMP директив).

Новыми свойствами, отличающими нашу систему от перечисленных здесь, являются ориентация на кластер и мультипроцессор и возможность пропустить взаимодействие с пользователем, лишь только запросив "добро" на вставку директив. Первое свойство особо важно, так как распараллеливание программ для будущих высокопроизводительных систем с многоуровневым параллелизмом связано с использованием обеих моделей OpenMP и DVM.

# Построение решения задачи

Так как решение задачи вставки OpenMP директив напрямую зависит от данных, предоставляемых анализатором, то построение решения задач "Эксперта" мы можем рассматривать только в тесной привязке к данным анализа.

## Формат входных данных

Работа с Базой Данных для эксперта происходит посредством интерфейса. Основная часть Базы Данных- дерево циклов. Структура дерева важна для описания алгоритмов работы эксперта. Вершины дерева могут быть следующих типов:

* Процедура. В данной версии "Системы автоматического распараллеливания" процедурой считается только тело программы.
* Цикл. Цикл содержит информацию обо всех переменных, использующихся в нем, зависимостях, найденных в нем анализатором, времени исполнения одного витка, количество итераций для данного варианта задачи.
* Линейный фрагмент.
* Условный оператор.

Выходными данные эксперта являются комментарии к вершинам дерева циклов. По этим комментариям в выходную программу вставляются директивы OpenMP. Важно заметить, что комментарии могут быть проставлены до вершины, что означает директивы OpenMP, соответствующие комментарию будут стоять до оператора, соответствующего вершине. Второй вариант – комментарии после вершины, что означает директивы OpenMP, соответствующие комментарию будут стоять после действий оператора, соответствующего вершине.

## Основные аспекты работы эксперта

При составлении алгоритмов эксперта были учтены следующие возможности OpenMP:

1. возможность работы с циклами (директива DO).
2. возможность обработки редукционных переменных (директива REDUCTION).
3. возможность исполнять часть кода внутри цикла в "естественном" порядке (директива ORDERED).
4. возможность приватизировать переменные (директива PRIVATE).
5. возможность образовывать конвейер посредством OpenMP директив (эта возможность будет рассмотрена ниже).
6. возможность отмены синхронизаций между параллельными циклами (директива NOWAIT).

При работе эксперта все циклы в программе разбиваются на **Потенциально Параллельные Циклы** (далее **ППЦ**) и **Циклы, Неподдающиеся Распараллеливанию** (далее **ЦНР**). ППЦ **–** цикл, который можно распараллелить средствами OpenMP, не обязательно эффективно. ЦНР **–** циклы, которые не поддаются распараллеливанию из-за зависимостей по данным между витками цикла. Такие циклы бывают следующих видов:

1. содержащие ввод/вывод.
2. содержащие выход из цикла.

Основная цель "Эксперта"- нахождение для последовательной программы набора директив OpenMP такого, что ускорение параллельной программы при заданном варианте работы программы будет максимально. Назовем такой набор директив **наилучшими комментариями** к программе. Все комментарии разобьем на два вида: комментарии по локализации переменных (PRIVATE, REDUCTION и т.д.) и все другие комментарии. Назовем **Вариантом параллелизма программы** (далее **Вариант параллелизма**) **–** вариантдиректив OpenMP, описывающих параллельные области и параллельные циклы (без директив локализации переменных). Тогда **Вариант локализации переменных** (**Вариант локализации**) **–** вариантдиректив локализации переменных. Для каждого варианта параллелизма может существовать несколько вариантов локализации. **Проверкой Альтернативной Локализации** назовемнахождение минимума оценочной функции для набора локализаций для данного варианта параллелизма с фиксированием комментариев, соответствующих минимальной оценочной функцией. **Схема распараллеливания –** фиксация одного варианта параллелизма и одного варианта локализации.

Работа Эксперта для мультипроцессора разделена на 4 стадии:

1. Разделение всех циклов программы на ППЦ и ЦНР, обозначение видов ППЦ и предварительное фиксирование набора вариантов локализации переменных.
2. Перебор всевозможных вариантов распараллеливания программы, предварительное фиксирование наилучшего варианта параллелизма.
3. Перебор вариантов локализаций программы, фиксирование наилучшей схемы распараллеливания, подсчет ускорения.
4. Расстановка директив OpenMP, соответствующих наилучшей схеме распараллеливания, в программу.

В процессе работы "Эксперта" создаются внутренние представления данных, обеспечивающие решение задачи, представленные в Таблице 1

|  |  |  |  |
| --- | --- | --- | --- |
| Название представления | На каком шаге создается | Что содержит | Зачем нужно |
| Первое внутреннее | 1 шаг | 1. Информация обо всех операторах программы  2. Информация о локализации программы "по умолчанию" | Базовое представление, на основании которого производится последующий анализ |
| Второе внутреннее | 2-3 шаг | Схема распараллеливания, которая считается наилучшей | Для формирования комментариев (содержит информацию по распараллеливанию) |
| Список параллельных регионов | 2 шаг | Все параллельные регионы программы | Для формирования комментариев (содержит информацию по локализации) |
| Программный список локализаций переменных | 3 шаг | Список переменных, для каждой из которых ведется список локализаций в параллельных регионах программы | Для простановки комментария threadprivate |

Таблица 1. Внутренние представления "Эксперта".

Первое внутреннее представление содержит информацию о локализации переменных.

Эта информация представляет собой 2 пары значений, по которым однозначно можно определить директиву OpenMP, с помощью которой переменная будет локализована в программе нужным способом.

Пометки проставляются для всех переменных, используемых внутри данного оператора. Все возможные пометки представлены в Таблице 2.

|  |  |
| --- | --- |
| 1-е значение 1-й пары (вид приватной) | 2-е значение 1-й пары - пометка приватной |
| private (по умолчанию)  firstprivate  lastprivate  reduction | Ind – индекс  Set – Необходимо указать в комментариях  Redtype() – тип редукции (только для редукционных)  Pos (по умолчанию) – возможно объявить приватной, но не обязательно  No – невозможно объявить приватной |
| 1-е значение 2-й пары (общая) | 2-е значение 2-й пары (пометка общей) |
| Shared (по умолчанию) | Def (по умолчанию) – общая по умолчанию  Set – необходимо указать в директивах  No – невозможно объявить общей |

Таблица 2. Пометки по локализации переменных.

Как пометки соответствуют директивам OpenMP:

Если 2-е значение 1-й пары = Set, переменную необходимо объявить PRIVATE.

Если 2-е значение 2-й пары = Set, переменную необходимо объявить SHARED.

Если 1-е значение 1-й пары = reduction, необходима директива REDUCTION(2-е значение 1-й пары: имя переменной).

Если 1-е значение 1-й пары = lastprivate, необходима директива LASTPRIVATE(имя переменной).

Если 1-е значение 1-й пары = firstprivate, необходима директива FIRSTPRIVATE(имя переменной).

## Оценочные функции

1. Оценочная функция стоимости параллельного региона вычисляется по следующей формуле:

**Стоимость параллельного исполнения** = (стоимость последовательного исполнения / число нитей + стоимость ordered) \* число итераций + стоимость создания и удаления параллельной области стоимость последовательного исполнения

Это стоимость одной итерации цикла (предоставлена анализатором) без учета стоимости выражений, окруженных директивами Ordered/End Ordered число нитей – количество "полезных" процессоров. Для цикла это значение равно минимуму из общего количества процессоров и количества итераций цикла.

стоимость ordered = стоимость последовательного исполнения выражений, окруженных директивами Ordered/End Ordered.

стоимость создания и удаления параллельной области = const + стоимость создания m приватных переменных + стоимость синхронизации. Константа зависит от архитектуры системы и берется из базы данных или определяется пользователем. Это время, необходимое для создания (инициализации) и удаления параллельного региона.

стоимость создания m приватных переменных = m, где m это количество приватных переменных в данном параллельном регионе. Так как известно, что единица стоимости последовательного исполнения программы эквивалентна единичной записи в память, то для создания m локальных переменных возьмем время равное m "условных единиц". Это связано с тем, что для каждой нити необходимо инициализировать m переменных.

стоимость синхронизации = количество приватных переменных. Синхронизация происходит при закрытии параллельного цикла. При этом необходимо синхронизировать все локальные копии переменных. Стоимость барьерной синхронизации, зависящей от аппаратуры, учитывается в стоимости удаления параллельной области.

1. Оценочная функция стоимости вычисляется по следующей формуле:

Стоимость конвейера = стоимость действий внутри конвейера \* количество действий конвейера + инициализация конвейера + синхронизация конвейера.

стоимость действий внутри конвейера = стоимость последовательного исполнения внутреннего цикла (конвейер возможен лишь для тесно вложенных циклов).

количество действий конвейера, количество действий, когда в работу задействована хотя бы 1 нить.

инициализация конвейера = const, зависит от архитектуры системы и берется из базы данных или определяется пользователем.

синхронизация конвейера = количество директив FLUSH, BARRIER и ENDDO \* количество локальных переменных \* количество действий конвейера.

1. Оценочная функция региона равна сумме всех оценочных функций членов региона.

# Пошаговая реализация "Эксперта"

## 

## Шаг 1. Предварительный анализ

**Данные на входе:** База Данных с результатами статического анализа.

**Данные на выходе:** 1-е внутреннее представление.

**Как проходит преобразование входных данных в выходные:**

Циклы разделяем на ППЦ и ЦНР, параллельно рассчитывая локализацию по умолчанию. Линейные фрагменты и условные операторы просто переносим. Проход по дереву осуществляется "сверху-вниз".

**Алгоритм преобразования входных данных в выходные:**

Проходим по дереву циклов из Базы Данных, в зависимости от вершины делаем соответствующие действия.

**Если вершина – цикл, неподдающийся распараллеливанию переносим его в первое внутреннее представление с пометкой ЦНР.**

**Если вершина – ППЦ, проводим следующую последовательность действий:**

1. В текущую версию комментариев к вершине в первом внутреннем представлении записать **ППЦ**.
2. Пометить уровень вложенности (исходя из того, что "тело" программы имеет уровень 0).
3. Для всех переменных, используемых в данном цикле во временном представлении сделать пометки:

* Для индекса: **private, ind, shared, no**
* Для остальных: **private, pos, shared, def**

1. Если есть редукция в цикле: для всех редукционных переменных сделать пометки: **reduction, "операция редукции", shared, no**
2. Еслив цикле присутствуетскалярная зависимость по данным: для всех переменных со скалярной зависимостью сделать пометки: **private (lastprivate), set,** **shared, no**
3. Еслицикл содержитзависимость по данным в массиве: проставить для данного цикла пометку **ordered** и для оператора, соответствующего первому использованию массива, имеющего зависимость, в теле цикла пометку "первое использование ordered", для оператора, соответствующего последнему использованию массива, имеющему зависимость, в теле цикла поставить пометку "последнее использование ordered". Массивы, вызвавшие зависимость занести в список "массивов с зависимостями".

**Если вершина – линейный фрагмент или ветвление - просто переносим вершину из Базы Данных в 1-е внутреннее представление.**

## Шаг 2. Выбор варианта распараллеливания

**Данные на входе:** База Данных с результатами статического анализа, 1-е внутреннее представление.

**Данные на выходе:** незаконченное 2-е внутреннее представление.

**Как проходит преобразование входных данных в выходные:**

Для ППЦ выбираем наилучший вариант комментариев. Одновременно создается 2-е внутреннее представление, отражающее наилучшую схему распараллеливания: перебором всех возможных вариантов распараллеливания и альтернативных локализаций. Поиск наилучшего распараллеливания происходит "снизу-вверх": сначала спускаемся на самый нижний уровень, находим распараллеливание для него, затем поднимаемся на 1 уровень выше и находим наилучшее представление для данного уровня и т.д. При нахождении наилучшего варианта распараллеливания на уровне пытаемся объединить подряд стоящие эффективные для распараллеливания циклы в параллельный регион. Если возникает конфликт локализации в регионе, он делится на регионы, в которых не будет конфликта.

На этом шаге обходим дерево из 1-ого внутреннего представления и выбираем наилучшую схему распараллеливания для вложенных ППЦ и объединяем в параллельные регионы "соседние" ППЦ. Для параллельных регионов перебираем все возможные варианты комментариев с фиксированием наилучшего из них во 2-м внутреннем представлении.

**Алгоритм преобразования входных данных в выходные:**

Проходим вершины дерева из 1-го внутреннего представления. Изначально все ППЦ помечаются как не пройденные. Продолжаем действия, пока не останется не пройденных ППЦ.

**Основные действия шага 2:**

1. Находим ППЦ наибольшего уровня (т.е. самый "низший" по вложенности) и делаем уровень найденного цикла текущим.
2. Для всех циклов текущего уровня сравниваем 3 оценочных функции:
3. оценочная функция для последовательного варианта,
4. оценочная функция при распараллеливании данного цикла,
5. оценочная функция при распараллеливании вложенных циклов, при том, что данный цикл остается последовательным (этот вариант оценивается только, если существуют вложенные ППЦ).
   1. Если минимальна функция a), то цикл помечается как неэффективный для распараллеливания. Если минимальна функция b), то цикл помечается как эффективный для распараллеливания и добавляется в параллельный регион. Вложенные циклы, признанные эффективными для распараллеливания становятся неэффективными для распараллеливания. Если минимальна функция с), то никаких действий не происходит.
   2. Циклы, для которых распараллеливание эффективно (минимальна функция b) добавляются в параллельный регион. Если на текущем уровне уже есть параллельный регион, причем последний пройденный цикл соседствует с текущим, то производится проверка на отсутствие конфликтов локализаций, если текущий цикл будет добавлен в параллельный регион. В случае отсутствия конфликтов, текущий цикл добавляется в регион. Если найдены конфликты локализации, или предыдущий цикл на уровне не соседствует с текущим, или текущий цикл первый на уровне, то для цикла создается новый параллельный регион. Если в параллельный регион добавляется цикл, то происходит проверка на установку Nowait.
   3. Далее, цикл помечается, как пройденный. Если на уровне не осталось не пройденных циклов, текущий уровень становится на единицу меньше, и продолжаются действия 2). Если же текущий уровень принял значение 0, то после того, как не останется не пройденных ППЦ формируется второе внутреннее представление, отражающее все циклы, эффективные для распараллеливания.

**Как происходит установка NOWAIT для подряд стоящих параллельных циклов:**

Условие проверяется для подряд идущих циклов при добавлении нового цикла в параллельный регион.

* Если у последнего цикла в параллельном регионе есть пометка NowaitIn (означает, что перед предыдущим циклом уже установлена директива NOWAIT), то проверить условие Nowait для пар (новый цикл, последний цикл в параллельном регионе). Если условие истинно проверять условие для пар всех (новый цикл, цикл с пометкой NowaitOut), причем начиная с предпоследнего и в обратном порядке, до первой неудачной проверки или до первого цикла без пометки NowaitOut. Если не было неудачных проверок, поставить у нового цикла пометку NowaitIn, а у последнего в параллельном регионе NowaitOut (означает, что после цикла установлена директива NOWAIT).
* Если же у последнего цикла в параллельном регионе нет пометки пометка NowaitIn, то проверить условие Nowait для пар (новый цикл, последний цикл в параллельном регионе). Если условие истинно, поставить у нового цикла пометку NowaitIn, а у последнего в параллельном регионе NowaitOut.

**Проверка условия Nowait между двумя циклами:**

* Если множество переменных/массивов для 2 подряд идущих циклов не пересекается (за исключением индексов - они могут быть одинаковыми), то проверка удачна.
* Иначе, сравнить индексы массивов. Для каждого обращения к неприватизованному массиву в левой части одного цикла и для каждого обращения к этому же массиву в другом цикле сравнить индексы. Если докажем для измерения, по которому идет распараллеливание, что индексы во всех обращениях различны и количество итераций в циклах равны, то проверка удачна. Иначе проверка неудачна.

Фактически проверенное условие представляет собой проверку, будет ли нить, выполняющая набор инструкций из первого цикла записывать/считывать данные, которые считаны/записаны другой нитью, выполняющей инструкции из другого цикла. Для каждого неприватизованного массива мы проверяем, что множество мест в памяти для записи у одной нити и множество мест чтения/записи другой не пересекаются.

**Для параллельных циклов с зависимостью по данным в массиве происходит проверка на возможность установки конвейера.**

Для распараллеливания циклов с зависимостью используется алгоритм, который был применен в тестах NASA. Этот алгоритм накладывает ряд ограничений, которые проверяются экспертом. Вариант конвейерной работы тестируется только, если цикл с конвейерной зависимостью является вложенным внутренним. В зависимости не участвуют "угловые" элементы. На Рис. 9 слева изображена зависимость без участия "угловых" элементов. То есть значение массива зависит от значения элементов при изменении лишь 1 измерения. В правой части Рис. 9 описан вариант зависимости, при котором установка конвейера невозможна из-за "угловых" элементов.

Можно

Нельзя

A[I,J]=A[I-1,J]+ A[I+1,J]+ A[I,J-1]+ A[I,J+1]

A[I,J]=A[I-1,J+1]

Рис. 9. Возможность установки конвейера.

Для такого цикла оценочная функция при распараллеливании цикла считается, как минимум оценочной функции:

1. для работы цикла с директивами ORDERED,
2. конвейерным распараллеливанием.

В случае если на основном шаге цикл был помечен, как эффективный для распараллеливания, то если минимальна функция a), то цикл помечается ORDERED, иначе PIPELINE.

## Шаг 3. Выбор варианта локализации

**Данные на входе:** База Данных с результатами статического анализа, 1-е внутреннее представление, незаконченное 2-е внутреннее представление.

**Данные на выходе:** 2-е внутреннее представление.

**Как проходит преобразование входных данных в выходные:**

Последний шаг по распараллеливанию: пытаемся выбрать наилучшую локализацию для текущего распараллеливания. Пробуем склеить соседние параллельные регионы за счет выбора подходящей локализации. Параллельно для массивов, у которых есть возможность установки **threadprivate**, создаем альтернативные локализации.

**Алгоритм преобразования входных данных в выходные:**

Для соседних параллельных регионов, находящихся на одном уровне пробуем альтернативные локализации. Если для какой—либо локализации регионы можно склеить, сравниваем оценочную функцию при склейке регионов и без склейки регионов. Если функция при склейке меньше, то превращаем 2 региона в один, помечаем для него локализацию.

После того, как не останется регионов, которые возможно склеить, внутри каждого региона проверяем все альтернативные локализации. Если оценочная функция для какой-либо альтернативной локализации меньше, чем оценочная функция для текущего варианта локализации, то эта альтернативная локализация заносится в текущий вариант локализации. Как только пройдены все регионы, информация из текущего варианта локализации заносится во 2-е внутреннее представление.

Если в Программном списке локализации переменных, какой-либо переменная во всех параллельных регионах становится приватным, то в программном списке локализаций переменных данный массив помечается как **threadprivate**. Для следующих параллельных регионов, использующих данный массив, он объявляется в списке **copyin**. Установка данных директив дает ускорение за счет отсутствия инициализации private переменных.

Для всех массивов, для которых не проставлены комментарии **threadprivate**, рассчитываем альтернативную локализации для всей программы, считая переменную приватной, но вычитая стоимость инициализации и синхронизации для массива. Если оценочная функция альтернативной локализации лучше оценочной функции для 2-го внутреннего представления, альтернативная локализация заносится во 2-е внутреннее представление, переменная во всех параллельных регионах становится: **private, def, shared, no** и помечается как **threadprivate.**

## Шаг 4. Внесение конечных комментариев в Базу Данных и подсчет ускорения

**Данные на входе:** База Данных, незаконченное 2-е внутреннее представление.

**Данные на выходе:** База Данных с комментариями.

**Как проходит преобразование входных данных в выходные:**

Из 2-го внутреннего представления комментарии переносятся в Базу Данных, соблюдая синтаксис OpenMP. Оценивается общее ускорение.

**Алгоритм преобразования входных данных в выходные:**

Проходим все вершины в Базе Данных. Каждую вершину пытаемся найти во 2-м внутреннем представлении, в случае успеха проставляем соответствующие комментарии следующим образом:

1. Если есть массивы с пометкой threadprivate, то в вершину с описанием переменных заносится **!$OMP THREADPRIVATE (список переменных).**
2. Если цикл первый в параллельном регионе и цикл выгодный для распараллеливания **!$OMP PARALLEL** <список необходимых к обозначению **PRIVATE** и **SHARED** переменных> <**COPYIN (список переменных)**>

<список необходимых к обозначению PRIVATE и SHARED переменных> - перечисление директив (клауз) вида PRIVATE (имя переменной) или SHARED (имя переменной). В одном списке может быть несколько и PRIVATE, и SHARED.

<**COPYIN (список переменных)**> - указывается, если в цикле используются переменныес пометкой **threadprivate**, причем все они должны быть перечислены через запятую в списке переменных.

1. Если цикл последний в параллельном регионе - **!$OMP END PARALLEL**, как директива после цикла.
2. Если цикл выгодный для распараллеливания - **!$OMP DO** <**ORDERED**> <список **REDUCTION, LASTPRIVATE, FIRSTPRIVATE**>

<**ORDERED**> - директива ORDERED. Указывается, если при распараллеливании цикла используется ORDERED.

<список **REDUCTION**, **LASTPRIVATE**, **FIRSTPRIVATE**> - перечисление директив (клауз) вида REDUCTION (имя переменной), или LASTPRIVATE (имя переменной), или FIRSTPRIVATE (имя переменной). В одном списке может быть несколько и REDUCTION, LASTPRIVATE, FIRSTPRIVATE.

1. Если цикл выгодный для распараллеливания - **!$OMP END DO**, как директива после цикла.
2. Если у цикла пометка Ordered, перед первой вершиной с пометкой "первое использование ORDERED" вносится **!$OMP ORDERED**.
3. Если у цикла пометка Ordered, в конец последней вершины с пометкой "последнее использование ORDERED" вносится, **!$OMP END ORDERED**.
4. Если у цикла пометка **pipeline** – вставляются следующие директивы:

В области инициализации переменных, проверяется уникальность и происходит объявление функций и переменных, необходимых для функционирования конвейера:

!$ INTEGER OMP\_GET\_NUM\_THREADS, OMP\_GET\_THREAD\_NUM

!$ INTEGER IAM, NUMT, ISYNC("количество процессоров")

При инициализации параллельного региона, в который входит цикл с конвейером:

!$OMP PARALLEL PRIVATE (IAM,NUMT,ILIMIT)

Перед do внешнего цикла - инициализация нитей и синхронизационного массива ISYNC:

!$ iam = omp\_get\_thread\_num ()

!$ numt = omp\_get\_num\_threads ()

!$ ILIMIT=MIN(NUMT-1,Число витков внешнего цикла)

!$ ISYNC (IAM) = 0

!$OMP BARRIER

До цикла с конвейерной зависимостью - инициализация конвейера, допуск к циклу для нитей только после того, как предыдущая нить сделала одну итерацию внешнего цикла и распараллеливание витков внутреннего цикла между нитями:

!$ IF (IAM .GT. 0 .AND. IAM .LE. ILIMIT) THEN

!$ DO WHILE (ISYNC (IAM-1) .EQ. 0)

!$OMP FLUSH (ISYNC)

!$ ENDDO

!$ ISYNC (IAM-1)=0

!$OMP FLUSH(ISYNC)

!$ ENDIF

!$OMP DO

Перед enddo внешнего цикла – дождаться, пока следующая нить запустится, и продолжать выполнение цикла:

!$OMP ENDDO NOWAIT

!$ IF (IAM .LT. ILIMIT) THEN

!$ DO WHILE (ISYNC (IAM) .EQ. 1)

!$OMP FLUSH (ISYNC)

!$ ENDDO

!$ ISYNC (IAM)=1

!$OMP FLUSH(ISYNC)

!$ ENDIF

После enddo внешнего цикла – синхронизация:

!$OMP BARRIER

Таким образом, все нити входят в параллельный регион. Отрабатывает 1-я нить со своей частью витков внутреннего цикла. Вступает в работу 2-я нить, работают обе нити, причем 1-я со 2-м значением итератора, а 2-я с первым и т.д. Пример функционирования конвейера показан на Рис. 10.

Внешний итератор

Разбиение

по

нитям

по внутреннему итератору

1(1)

1(2)

2(2)

1(3)

2(3)

3(3)

3(4)

2(4)

3(5)

…

…

Рис. 10. Вычисление цикла с конвейерной зависимостью для 2-х мерного случая. Квадраты – области элементов массива; цифра - № нити, которая вычислит эту область; цифра в скобках – шаг работы конвейера, на котором вычислится область.

## Примеры работы алгоритма

### Программа "Якоби"

1. **Листинг последовательной программы**

PROGRAM JAC

PARAMETER (L=8, ITMAX=20)

REAL A(L,L), EPS, MAXEPS, B(L,L)

PRINT \*, '\*\*\*\*\*\*\*\*\*\* TEST\_JACOBI \*\*\*\*\*\*\*\*\*\*'

MAXEPS = 0.5E - 7

DO 1 J = 1, L(1)

DO 1 I = 1, L(2)

A(I, J) = 0. IF(I.EQ.1 .OR. J.EQ.1 .OR. I.EQ.L .OR. J.EQ.L) THEN

B(I, J) = 0.

ELSE

B(I, J) = ( 1. + I + J ) ENDIF

1 CONTINUE

DO 2 IT = 1, ITMAX(3)

EPS = 0.

DO 21 J = 2, L-1(4)

DO 21 I = 2, L-1(5)

EPS = MAX ( EPS, ABS( B( I, J) - A( I, J)))

A(I, J) = B(I, J)

21 CONTINUE

DO 22 J = 2, L-1(6)

DO 22 I = 2, L-1(7)

B(I, J) = (A( I-1, J ) + A( I, J-1 ) + A( I+1, J)+ A( I, J+1 )) / 4

22 CONTINUE

PRINT 200, IT, EPS

200 FORMAT(' IT = ',I4, ' EPS = ', E14.7)

IF ( EPS . LT . MAXEPS ) GO TO 3

2 CONTINUE

3 OPEN (3, FILE='JAC.DAT', FORM='FORMATTED', STATUS='UNKNOWN')

WRITE (3,\*) B CLOSE (3) END

1. **Первое внутреннее представление для циклов**

После первого шага в первом внутреннем представлении помеченным в листинге последовательной программы циклам будут соответствовать следующие вершины:

(1) Тип:ППЦ,уровень = 0, число итераций = 8

Использование переменных:

Имя: i. Пометки: PRIVATE, SET, SHARED, NO

Имя: j. Пометки: PRIVATE, IND, SHARED, NO

Имя: a. Пометки: PRIVATE, POS, SHARED, DEF

Имя: b. Пометки: PRIVATE, POS, SHARED, DEF

(2) Тип:ППЦ,уровень = 1, число итераций = 8

Использование переменных:

Имя: i. Пометки: PRIVATE, IND, SHARED, NO

Имя: j. Пометки: PRIVATE, POS, SHARED, DEF

Имя: a. Пометки: PRIVATE, POS, SHARED, DEF

Имя: b. Пометки: PRIVATE, POS, SHARED, DEF

(3) Тип:ЦНР,уровень = 0, число итераций = 20

Использование переменных:

Имя: eps. Пометки: PRIVATE, SET, SHARED, NO

Имя: j. Пометки: PRIVATE, SET, SHARED, NO

Имя: i. Пометки: PRIVATE, SET, SHARED, NO

Имя: b. Пометки: PRIVATE, NO, SHARED, SET ORDERED!

Имя: a. Пометки: PRIVATE, NO, SHARED, SET ORDERED!

(4) Тип:ППЦ,уровень = 1, число итераций = 6

Использование переменных:

Имя: i. Пометки: PRIVATE, SET, SHARED, NO

Имя: eps. Пометки: REDUCTION, MAX, SHARED, NO

Имя: b. Пометки: PRIVATE, POS, SHARED, DEF

Имя: j. Пометки: PRIVATE, IND, SHARED, NO

Имя: a. Пометки: PRIVATE, POS, SHARED, DEF

(5) Тип:ППЦ,уровень = 2, число итераций = 6

Использование переменных:

Имя: eps. Пометки: REDUCTION, MAX, SHARED, NO

Имя: b. Пометки: PRIVATE, POS, SHARED, DEF

Имя: i. Пометки: PRIVATE, IND, SHARED, NO

Имя: j. Пометки: PRIVATE, POS, SHARED, DEF

Имя: a. Пометки: PRIVATE, POS, SHARED, DEF

(6) Тип:ППЦ,уровень = 1, число итераций = 6

Использование переменных:

Имя: i. Пометки: PRIVATE, SET, SHARED, NO

Имя: a. Пометки: PRIVATE, POS, SHARED, DEF

Имя: j. Пометки: PRIVATE, IND, SHARED, NO

Имя: b. Пометки: PRIVATE, POS, SHARED, DEF

(7) Тип:ППЦ,уровень = 2, число итераций = 6

Использование переменных:

Имя: a. Пометки: PRIVATE, POS, SHARED, DEF

Имя: i. Пометки: PRIVATE, IND, SHARED, NO

Имя: j. Пометки: PRIVATE, POS, SHARED, DEF

Имя: b. Пометки: PRIVATE, POS, SHARED, DEF

**3) Второй шаг**

На этом шаге будет устроен перебор различных вариантов распараллеливания. Рассмотрим их, группируя по регионам. Все варианты будем рассматривать при числе процессоров=4. 1-й регион: циклы (1) и (2). Здесь будет рассмотрено 3 варианта, время исполнения 1 витка внутреннего цикла =6, L=8.

|  |  |
| --- | --- |
| !$OMP PARALLEL PRIVATE (I)  !$OMP DO  DO 1 J = 1, L  DO 1 I = 1, L  A(I, J) = 0.  IF(I.EQ.1 .OR. J.EQ.1 .OR. I.EQ.L .OR. J.EQ.L) THEN  B(I, J) = 0.  ELSE  B(I, J) = ( 1. + I + J )  ENDIF  ENDDO  ENDDO  !$OMP END DO  !$OMP END PARALLEL | DO 1 J = 1, L  !$OMP PARALLEL  !$OMP DO  DO 1 I = 1, L  A(I, J) = 0.  IF(I.EQ.1 .OR. J.EQ.1 .OR. I.EQ.L .OR. J.EQ.L) THEN  B(I, J) = 0.  ELSE  B(I, J) = ( 1. + I + J )  ENDIF  ENDDO  !$OMP END DO  ENDDO  !$OMP END PARALLEL |
| Стоимость = 6\*8\*8/4 + 14 = 110  Количество приватных = 2  Время инициализации = 5  Время синхронизации переменных и удаления региона = 7  Итого: 14 | Стоимость = 8\*(6\*8/4 + 12) = 192  Количество приватных = 1  Время инициализации = 5  Время синхронизации переменных и удаления региона = 6  Итого: 12 |

Рис. 11. Сравнение вариантов распараллеливания в 1-м регионе программы "Якоби".

На Рис. 11 показаны возможные рассматриваемые варианты распараллеливания: цикла по J в левой части и цикла по I в левой части. Пункт "Стоимость" отражает подсчет оценочной функции. Она берется, как сумма "параллельного" вычисления цикла и "дополнительных" расходов, подсчитанных в пункте "Итого". Так как стоимость последовательная = 6\*8\*8 = 384, будет выбран вариант, показанный в левой части.

2-й регион: циклы (4) и (5). Время исполнения 1 витка внутреннего цикла =5.

|  |  |
| --- | --- |
| !$OMP PARALLEL PRIVATE (I)  !$OMP DO REDUCTION(EPS,MAX)  DO 21 J = 2, L-1  DO 21 I = 2, L-1  EPS = MAX ( EPS, ABS( B( I, J) - A( I, J)))  A(I, J) = B(I, J)  21 CONTINUE  !$OMP END DO  !$OMP END PARALLEL | DO 21 J = 2, L-1  !$OMP PARALLEL  !$OMP DO REDUCTION(EPS,MAX)  DO 21 I = 2, L-1  EPS = MAX ( EPS, ABS( B( I, J) - A( I, J)))  A(I, J) = B(I, J)  !$OMP END DO  !$OMP END PARALLEL  21 CONTINUE |
| Стоимость = 5\*6\*6/4+16=61  Количество приватных = 3  Время инициализации = 5  Время синхронизации переменных и удаления региона = 8  Итого: 16 | Стоимость = 6\*(5\*6/4+14)=129  Количество приватных = 2  Время инициализации = 5  Время синхронизации переменных и удаления региона = 7  Итого: 14 |

Рис. 12. Сравнение вариантов распараллеливания в 2-м регионе программы "Якоби".

На Рис. 12 показаны возможные рассматриваемые варианты распараллеливания: цикла по J в левой части и цикла по I в левой части. Стоимость последовательная = 6\*6\*5=**180**, следовательно, будет выбран вариант, описанный в левой части Рис. 12. Редукционная переменная считается как приватная.

3-й регион: циклы (6) и (7). Время исполнения 1 витка внутреннего цикла =9.

|  |  |
| --- | --- |
| !$OMP PARALLEL PRIVATE (I)  !$OMP DO  DO 22 J = 2, L-1  DO 22 I = 2, L-1  B(I, J) = (A( I-1, J ) + A( I, J-1 ) + A( I+1, J)+ A( I, J+1 )) / 4  22 CONTINUE  !$OMP END DO  !$OMP END PARALLEL | DO 21 J = 2, L-1  !$OMP PARALLEL  !$OMP DO SHARED(J), REDUCTION(EPS,MAX)  DO 21 I = 2, L-1  EPS = MAX ( EPS, ABS( B( I, J) - A( I, J)))  A(I, J) = B(I, J)  !$OMP END DO  !$OMP END PARALLEL  21 CONTINUE |
| Стоимость = 9\*6\*6/4+14=95  Количество приватных = 2  Время инициализации = 5  Время синхронизации переменных и удаления региона = 7  Итого: 14 | Стоимость = 6\*(9\*6/4+12)=153  Количество приватных = 1  Время инициализации = 5  Время синхронизации переменных и удаления региона = 7  Итого: 12 |

Рис. 13. Сравнение вариантов распараллеливания в 3-м регионе программы "Якоби".

На Рис. 13 показаны возможные рассматриваемые варианты распараллеливания: цикла по J в левой части и цикла по I в левой части. Стоимость последовательная = 9\*6\*6=324, следовательно, будет выбран 1-й вариант из Рис. 13.

2-й и 3-й параллельные регионы объединятся в один регион, т.к. нет противоречий по локализации переменных. Проверка на NOWAIT не будет успешна: для обращения **A(I, J) = B(I, J)** есть обращение **B(I, J) = (A( I-1, J ) + A( I, J-1 ) + A( I+1, J)+ A( I, J+1 )) / 4** (может возникнуть ситуация, когда одной нити придется использовать "старое" значение массива A во втором цикле).

1. **Шаг 3 и Шаг 4**

На шаге 3 не найдется параллельных соседних параллельных регионов для склейки. Из всевозможных альтернативных локализаций ни одна не будет принята, т.к. в любой из них количество приватных переменных (а следовательно и оценочная функция будут расти). Для простановки threadprivate не найдется ни одного приватного массива.

На шаге 4 получим выходную программу:

program jac

parameter (l = 8,itmax = 20)

real a(l,l),eps,maxeps,b(l,l)

! arrays A and B with block distribution

print \*, '\*\*\*\*\*\*\*\*\*\* TEST\_JACOBI \*\*\*\*\*\*\*\*\*\*'

maxeps = 5.000000e-008

!nest of two parallel loops, iteration (i,j) will be executed on

!processor, which is owner of element A(i,j)

**!OMP PARALLEL PRIVATE(i)**

**!OMP DO**

do j = 1,l

do i = 1,l

a(i,j) = 0.

if (i .eq. 1 .or. j .eq. 1 .or. i .eq. l .or. j .eq. l) then

b(i,j) = 0.

else

b(i,j) = 1. + i + j

endif

enddo

enddo

**!OMP ENDDO**

**!OMP END PARALLEL**

do it = 1,itmax

eps = 0

!variable EPS is used for calculation of maximum value

**!OMP PARALLEL PRIVATE(i)**

**!OMP DO REDUCTION(MAX:eps)**

do j = 2,l - 1

do i = 2,l - 1

eps = max (eps,abs (b(i,j) - a(i,j)))

a(i,j) = b(i,j)

enddo

enddo

!Copying shadow elements of array A from

!neighbouring processors before loop execution

**!OMP ENDDO**

**!OMP DO**

do j = 2,l - 1

do i = 2,l - 1

b(i,j) = (a(i - 1,j) + a(i,j - 1) + a(i + 1,j) + a(i,j +

&1)) / 4

enddo

enddo

**!OMP ENDDO**

**!OMP END PARALLEL**

print 200, it,eps

200 FORMAT(' IT = ',I4, ' EPS = ', E14.7)

if (eps .lt. 5.000000e-008) goto 3

enddo

3 open (unit = 3,file = 'JAC.DAT',form = 'FORMATTED',status = 'UNKNO

&WN')

write (unit = 3,fmt = \*) b

close (unit = 3)

end

Для данной программы будет выдана следующая оценка производительности:

**Общее время последовательного выполнения: 10547 у.е.**

**Общее время параллельного выполнения (4 нити): 3231 у.е**

**Суммарное ускорение: в 3,26 раза**

### Программа "Sor"

1. **Листинг последовательной программы**

PROGRAM SOR

PARAMETER ( N = 10 )

REAL A( N, N ), EPS, MAXEPS, W

INTEGER ITMAX

PRINT \*, '\*\*\*\*\*\*\*\*\*\* TEST\_SOR \*\*\*\*\*\*\*\*\*\*'

ITMAX=20

MAXEPS = 0.5E - 5

W = 0.5

DO 1 J = 1, N

DO 1 I = 1, N

IF ( I .EQ.J) THEN

A( I, J ) = N + 2

ELSE

A( I, J ) = -1.0

ENDIF

1CONTINUE

DO 2 IT = 1, ITMAX

EPS = 0.

DO 21 J = 2, N-1

DO 21 I = 2, N-1

S = A( I, J )

A( I, J ) = (W / 4) \* (A( I-1, J ) + A( I+1, J ) + A( I, J-1 ) +

\*A( I, J+1 )) + ( 1-W ) \* A( I, J)

EPS = MAX ( EPS, ABS( S - A( I, J )))

21CONTINUE PRINT 200, IT, EPS

200 FORMAT(' IT = ',I4, ' EPS = ', E14.7)

IF (EPS .LT. MAXEPS ) GO TO 4

2CONTINUE

4 OPEN (3, FILE='SOR.DAT', FORM='FORMATTED',STATUS='UNKNOWN')

WRITE (3,\*) A CLOSE (3) END

1. **Первое внутреннее представление для циклов**

После первого шага в первом внутреннем представлении помеченным в листинге последовательной программы циклам будут соответствовать следующие вершины:

(1) Тип:ППЦ,уровень = 0, число итераций = 8

Использование переменных:

Имя: i. Пометки: PRIVATE, SET, SHARED, NO

Имя: j. Пометки: PRIVATE, IND, SHARED, NO

Имя: a. Пометки: PRIVATE, POS, SHARED, DEF

(2) Тип:ППЦ,уровень = 1, число итераций = 8

Использование переменных:

Имя: i. Пометки: PRIVATE, IND, SHARED, NO

Имя: j. Пометки: PRIVATE, POS, SHARED, DEF

Имя: a. Пометки: PRIVATE, POS, SHARED, DEF

(3) Тип:ЦНР,уровень = 0, число итераций = 20

Использование переменных:

Имя: eps. Пометки: PRIVATE, SET, SHARED, NO

Имя: j. Пометки: PRIVATE, SET, SHARED, NO

Имя: i. Пометки: PRIVATE, SET, SHARED, NO

Имя: a. Пометки: PRIVATE, NO, SHARED, SET ORDERED! PIPELINE!

Имя: s. Пометки: PRIVATE, SET, SHARED, NO

(4) Тип:ППЦ,уровень = 1, число итераций = 6

Использование переменных:

Имя: i. Пометки: PRIVATE, SET, SHARED, NO

Имя: eps. Пометки: REDUCTION, MAX, SHARED, NO

Имя: j. Пометки: PRIVATE, IND, SHARED, NO

Имя: a. Пометки: PRIVATE, NO, SHARED, SET, rw= 2 ORDERED! PIPELINE!

Имя: s. Пометки: PRIVATE, SET, SHARED, NO

(5) Тип:ППЦ,уровень = 2, число итераций = 6

Использование переменных:

Имя: i. Пометки: PRIVATE, IND, SHARED, NO

Имя: j. Пометки: PRIVATE, POS, SHARED, DEF

Имя: eps. Пометки: REDUCTION, MAX, SHARED, NO

Имя: a. Пометки: PRIVATE, NO, SHARED, SET, rw= 2 ORDERED! PIPELINE!

Имя: s. Пометки: PRIVATE, SET, SHARED, NO

1. **Второй шаг**

На этом шаге будет устроен перебор различных вариантов распараллеливания. Рассмотрим их, группируя по регионам. Все варианты будем рассматривать при числе процессоров=4.

1-й регион: циклы (1) и (2). Здесь будет рассмотрено 3 варианта, время последовательного исполнения 1 витка = 1.

|  |  |
| --- | --- |
| !$OMP PARALLEL PRIVATE(I)  !$OMP DO  DO 1 J = 1, N  DO 1 I = 1, N  IF ( I .EQ.J) THEN  A( I, J ) = N + 2  ELSE  A( I, J ) = -1.0  ENDIF  1CONTINUE  !$OMP END DO  !$OMP END PARALLEL | DO 1 J = 1, N  !$OMP PARALLEL  !$OMP DO  DO 1 I = 1, N  IF ( I .EQ.J) THEN  A( I, J ) = N + 2  ELSE  A( I, J ) = -1.0  ENDIF  !$OMP END DO  !$OMP END PARALLEL  1 CONTINUE |
| Стоимость = 1\*10\*10/4 + 14 = 39  Количество приватных = 2  Время инициализации = 5  Время синхронизации переменных и удаления региона = 7  Итого: 14 | Стоимость = 10 \* (1\*10/ 4 + 12) = 145  Количество приватных = 1  Время инициализации = 5  Время синхронизации переменных и удаления региона = 6  Итого: 12 |

Рис. 14. Сравнение вариантов распараллеливания в 1-м регионе программы "SOR".

На Рис. 14 показаны возможные рассматриваемые варианты распараллеливания: цикла по J в левой части и цикла по I в левой части. Стоимость последовательная = 10\*10=100, следовательно, будет выбран вариант, показанный в левой части Рис. 14.

2-й регион: циклы (4) и (5). Здесь будет рассмотрено 4 варианта, время последовательного исполнения 1 витка = 22.

|  |  |
| --- | --- |
| !$OMP PARALLEL PRIVATE(S) PRIVATE(I)  !$OMP DO REDUCTION(EPS,MAX)  DO 21 J = 2, N-1  DO 21 I = 2, N-1  !$OMP ORDERED  S = A( I, J )  A( I, J ) = (W / 4) \* (A( I-1, J ) + A( I+1, J ) + A( I, J-1 ) +  \*A( I, J+1 )) + ( 1-W ) \* A( I, J)  EPS = MAX ( EPS, ABS( S - A( I, J )))  !$OMP END ORDERED  21CONTINUE  !$OMP END DO  !$OMP END PARALLEL | DO 21 J = 2, N-1  !$OMP PARALLEL PRIVATE(S)  !$OMP DO REDUCTION(EPS,MAX)  DO 21 I = 2, N-1  !$OMP ORDERED  S = A( I, J )  A( I, J ) = (W / 4) \* (A( I-1, J ) + A( I+1, J ) + A( I, J-1 ) +  \*A( I, J+1 )) + ( 1-W ) \* A( I, J)  EPS = MAX ( EPS, ABS( S - A( I, J ))) !$OMP END ORDERED  !$OMP END DO  !$OMP END PARALLEL  21 CONTINUE |
| Стоимость = 22\*8\*8 + 18 = 1426  Количество приватных = 4  Время инициализации = 5  Время синхронизации переменных и удаления региона = 9  Итого: 18 | Стоимость = 8\*(22\*8 + 16) = 1536  Количество приватных = 3  Время инициализации = 5  Время синхронизации переменных и удаления региона = 8  Итого: 16 |

Рис. 15. Сравнение вариантов распараллеливания в 2-м регионе программы "SOR" с директивой ORDERED.

|  |
| --- |
| !$OMP PARALLEL  !$ iam = omp\_get\_thread\_num ()  !$ numt = omp\_get\_num\_threads ()  !$ ISYNC (IAM) = 0  !$ ILIMIT=MIN(NUMT-1,N-1-2)  !$OMP BARRIER  DO 21 J = 2, N-1  !$OMP DO PRIVATE(S), REDUCTION(EPS,MAX)  !$ IF (IAM .GT. 0 .AND. IAM .LE. ILIMIT) THEN  !$ DO WHILE (ISYNC(IAM-1) .EQ. 0)  !$OMP FLUSH(ISYNC)  !$ ENDDO  !$ ISYNC(IAM-1)=0  !$OMP FLUSH(ISYNC)  !$ ENDIF  DO 21 I = 2, N-1  S = A( I, J )  A( I, J ) = (W / 4) \* (A( I-1, J ) + A( I+1, J ) + A( I, J-1 ) +  \*A( I, J+1 )) + ( 1-W ) \* A( I, J)  EPS = MAX ( EPS, ABS( S - A( I, J )))  !$OMP ENDDO NOWAIT  !$ IF (IAM .LT. ILIMIT) THEN  !$ DO WHILE (ISYNC (IAM) .EQ. 1)  !$OMP FLUSH (ISYNC)  !$ ENDDO  !$ ISYNC (IAM)=1  !$OMP FLUSH(ISYNC)  !$ ENDIF  !$OMP END DO  !$OMP BARRIER  !$OMP END PARALLEL  21 CONTINUE |
| Стоимость = 22\*32+3+9+22 = 738  Количество действий конвейера = 32  Количество приватных = 3  Время инициализации конвейера без учета приватных = 5 + 4 = 9  Время синхронизации переменных и удаления региона = 3+5+4+3+3+4 = 22 (синхронизация приватных + инициализация региона + барьер до первого цикла + по барьеру во время инициализации каждой нити + по барьеру во время инициализации последующей нити + барьер в конце региона) |

Рис. 16. Схема распараллеливания в 2-м регионе программы "SOR" при конвейерном варианте.

На Рис. 15 и Рис. 16 показаны варианты параллелизма, которые будут перебираться "Экспертом". Стоимость последовательная = 22\*8\*8=1408, следовательно, будет выбран вариант c конвейером.

1. **Шаг 3 и Шаг 4**

На шаге 3 не найдется параллельных соседних параллельных регионов для склейки. Из всевозможных альтернативных локализаций ни одна не будет принята, т.к. в любой из них количество приватных переменных (а следовательно, и оценочная функция будут расти). Для простановки threadprivate не найдется ни одного приватного массива.

На шаге 4 получим выходную программу для варианта задачи с 4 процессорами:

program sor

parameter (n = 10)

real a(n,n),eps,maxeps,w

integer itmax

!$INTEGER OMP\_GET\_NUM\_THREADS,OMP\_GET\_THREAD\_NUM

!$INTEGER IAM, NUMT,ISYNC(4)

print \*, '\*\*\*\*\*\*\*\*\*\* TEST\_SOR \*\*\*\*\*\*\*\*\*\*'

itmax = 20

maxeps = 5.000000e-006

w = 5.000000e-001

!$OMP PARALLEL PRIVATE(i)

!$OMP DO

do j = 1,n

do i = 1,n

if (i .eq. j) then

a(i,j) = n + 2

else

a(i,j) = (-(1.0))

endif

enddo

enddo

!$OMP END DO

!$OMP END PARALLEL

do it = 1,20

eps = 0

!$OMP PARALLEL PRIVATE (IAM,NUMT,ILIMIT) PRIVATE(i) SHARED(a) PRIVATE(s) & &REDUCTION(MAX:eps)

!$ iam = omp\_get\_thread\_num ()

!$ numt = omp\_get\_num\_threads ()

!$ ISYNC (IAM) = 0

!$ ILIMIT=min(NUMT-1,n-1-2)

!$OMP BARRIER

do j = 2,n - 1

!$IF (IAM .GT. 0 .AND. IAM .LE. ILIMIT) THEN

!$ DO WHILE (ISYNC(IAM-1) .EQ. 0)

!$OMP FLUSH(ISYNC)

!$ ENDDO

!$ ISYNC(IAM-1)=0

!$OMP FLUSH(ISYNC)

!$ ENDIF

!$OMP DO

do i = 2,n - 1

s = a(i,j)

a(i,j) = 5.000000e-001 / 4 \* (a(i - 1,j) + a(i + 1,j) + a

&(i,j - 1) + a(i,j + 1)) + (1 - 5.000000e-001) \* a(i,j)

eps = max (eps,abs (s - a(i,j)))

enddo

!$OMP ENDDO NOWAIT

!$ IF (IAM .LT. ILIMIT) THEN

!$ DO WHILE (ISYNC (IAM) .EQ. 1)

!$OMP FLUSH (ISYNC)

!$ ENDDO

!$ ISYNC (IAM)=1

!$OMP FLUSH(ISYNC)

!$ ENDIF

enddo

!$OMP BARRIER

!$OMP END PARALLEL

print 200, it,eps

200 FORMAT(' IT = ',I4, ' EPS = ', E14.7)

if (eps .lt. 5.000000e-006) goto 4

enddo

4 open (unit = 3,file = 'SOR.DAT',form = 'FORMATTED',status = 'UNKNO

&WN')

write (unit = 3,fmt = \*) a

close (unit = 3)

end

Для данной программы будет выдана следующая оценка производительности:

**Общее время последовательного выполнения: 28343 у.е.**

**Общее время параллельного выполнения (4 нити): 14880 у.е.**

**Суммарное ускорение: в 1,9 раза**

### Программа "Модифицированный SOR"

Данный пример приведен для того, чтобы показать работу "Эксперта" на зависимостях по данным в массиве, для которого невозможно применение конвейера.

1. **Листинг последовательной программы**

PROGRAM SOR

PARAMETER ( N = 10 )

REAL A( N, N ), B(N,N), EPS, MAXEPS, W

INTEGER ITMAX

PRINT \*, '\*\*\*\*\*\*\*\*\*\* TEST\_SOR \*\*\*\*\*\*\*\*\*\*'

ITMAX=20

MAXEPS = 0.5E - 5

W = 0.5

DO 1 J = 1, N

DO 1 I = 1, N

IF ( I .EQ.J) THEN

A( I, J ) = N + 2

ELSE

A( I, J ) = -1.0

ENDIF

1CONTINUE

DO 2 IT = 1, ITMAX

EPS = 0.

DO 21 J = 2, N-1

DO 21 I = 2, N-1

S = A(I,J)

A( I, J ) = (W / 4) \* (A( I-1, J ) + A( I+1, J ) + A( I, J-1 ) +

\*A( I, J+1 )) + ( 1-W ) \* A( I+1, J+1)

B(I,J) = A(I,J)

EPS = MAX ( EPS, ABS( S - B( I, J )))

21CONTINUE

PRINT 200, IT, EPS

200 FORMAT(' IT = ',I4, ' EPS = ', E14.7)

IF (EPS .LT. MAXEPS ) GO TO 4

2CONTINUE

4 OPEN (3, FILE='SOR.DAT', FORM='FORMATTED',STATUS='UNKNOWN')

WRITE (3,\*) A

CLOSE (3)

END

1. **Работа "Эксперта"**

Для данной программы будут произведены действия, аналогичные тем, что описаны в пункте 5.5.2. Остановимся лишь на отличиях в шаге 2 и выходных данных.

2-й регион: циклы (4) и (5). Здесь будет рассмотрено 3 варианта, время последовательного исполнения первых 3 директив = 19, 4-й директивы = 4:

|  |  |
| --- | --- |
| !$OMP PARALLEL PRIVATE(S) PRIVATE(I)  !$OMP DO REDUCTION(EPS,MAX)  DO 21 J = 2, N-1  DO 21 I = 2, N-1  !$OMP ORDERED  S = A(I,J)  A( I, J ) = (W / 4) \* (A( I-1, J ) + A( I+1, J ) + A( I, J-1 ) +  \*A( I, J+1 )) + ( 1-W ) \* A( I+1, J+1)  B(I,J) = A(I,J)  !$OMP END ORDERED  EPS = MAX ( EPS, ABS( S - B( I, J )))  21CONTINUE  !$OMP END DO  !$OMP END PARALLEL | DO 21 J = 2, N-1  !$OMP PARALLEL PRIVATE(S)  !$OMP DO REDUCTION(EPS,MAX)  DO 21 I = 2, N-1  !$OMP ORDERED  S = A(I,J)  A( I, J ) = (W / 4) \* (A( I-1, J ) + A( I+1, J ) + A( I, J-1 ) +  \*A( I, J+1 )) + ( 1-W ) \* A( I+1, J+1)  B(I,J) = A(I,J)  !$OMP END ORDERED  EPS = MAX ( EPS, ABS( S - B( I, J )))  !$OMP END DO  !$OMP END PARALLEL  21 CONTINUE |
| Стоимость = 19\*8\*8 + 4\*8\*8/4 + 18 = 1298  Количество приватных = 4  Время инициализации = 5  Время синхронизации переменных и удаления региона = 9  Итого: 18 | Стоимость = 8\*(19\*8 + 4\*8/4 + 16) = 1408  Количество приватных = 3  Время инициализации = 5  Время синхронизации переменных и удаления региона = 8  Итого: 16 |

Рис. 17. Сравнение вариантов распараллеливания в 2-м регионе программы "модифицированный SOR".

На Рис. 17 показаны возможные варианты параллелизма для данного региона. Как видно, случай с конвейером не будет рассмотрен, из-за того, что элемент A(I,J) зависит от "бокового" элемента A(I+1,J+1). Стоимость последовательная = 23\*8\*8=1472, следовательно, будет выбран 1-й вариант.

На выходе получим следующую программу:

program sor

parameter (n = 10)

real a(n,n),b(n,n),eps,maxeps,w

integer itmax

print \*, '\*\*\*\*\*\*\*\*\*\* TEST\_SOR \*\*\*\*\*\*\*\*\*\*'

itmax = 20

maxeps = 5.000000e-006

w = 5.000000e-001

!$OMP PARALLEL PRIVATE(i)

!$OMP DO

do j = 1,n

do i = 1,n

if (i .eq. j) then

a(i,j) = n + 2

else

a(i,j) = (-(1.0))

endif

enddo

enddo

!$OMP END DO

!$OMP END PARALLEL

do it = 1,20

eps = 0

!$OMP PARALLEL PRIVATE(i) SHARED(a) PRIVATE(s)

!$OMP DO ORDERED REDUCTION(MAX:eps)

do j = 2,n - 1

do i = 2,n - 1

!$OMP ORDERED

s = a(i,j)

a(i,j) = 5.000000e-001 / 4 \* (a(i - 1,j) + a(i + 1,j) + a

&(i,j - 1) + a(i,j + 1)) + (1 - 5.000000e-001) \* a(i + 1,j + 1)

b(i,j) = a(i,j)

!$OMP END ORDERED

eps = max (eps,abs (s - b(i,j)))

enddo

enddo

!$OMP END DO

!$OMP END PARALLEL

print 200, it,eps

200 FORMAT(' IT = ',I4, ' EPS = ', E14.7)

if (eps .lt. 5.000000e-006) goto 4

enddo

4 open (unit = 3,file = 'SOR.DAT',form = 'FORMATTED',status = 'UNKNO

&WN')

write (unit = 3,fmt = \*) a

close (unit = 3)

end

**Общее время последовательного выполнения: 29623 у.е.**

**Общее время параллельного выполнения (4 нити): 26143 у.е.**

**Суммарное ускорение: в 1,13 раза**

### Программа "Модифицированный Якоби"

Данный пример приведен для того, чтобы показать работу "Эксперта", при установке директивы NOWAIT. Исходная программа "Якоби" была изменена, чтобы проверка установки NOWAIT была успешна. Работа "Эксперта" на этом примере повторяет действия из пункта 5.5.1 за исключением того, что проверка установки NOWAIT даст положительный результат.

1. **Листинг последовательной программы**

PROGRAM JAC

PARAMETER (L=8, ITMAX=20)

REAL A(L,L), EPS, MAXEPS, B(L,L), C(L,L)

C arrays A and B with block distribution

PRINT \*, '\*\*\*\*\*\*\*\*\*\* TEST\_JACOBI \*\*\*\*\*\*\*\*\*\*'

MAXEPS = 0.5E - 7

Cnest of two parallel loops, iteration (i,j) will be executed on

Cprocessor, which is owner of element A(i,j)

DO 1 J = 1, L

DO 1 I = 1, L

A(I, J) = 0.

IF(I.EQ.1 .OR. J.EQ.1 .OR. I.EQ.L .OR. J.EQ.L) THEN

B(I, J) = 0.

ELSE

B(I, J) = ( 1. + I + J )

ENDIF

1 CONTINUE

DO 2 IT = 1, ITMAX

EPS = 0.

Cvariable EPS is used for calculation of maximum value

DO 21 J = 2, L-1

DO 21 I = 2, L-1

A(I, J) = B(I, J)

21 CONTINUE

CCopying shadow elements of array A from

Cneighbouring processors before loop execution

DO 22 J = 2, L-1

DO 22 I = 2, L-1

A(I, J) = (C( I-1, J ) + C( I, J-1 ) + C( I+1, J)+

\* C( I, J+1 )) / 4

22 CONTINUE

PRINT 200, IT, EPS

200 FORMAT(' IT = ',I4, ' EPS = ', E14.7)

IF ( EPS . LT . MAXEPS ) GO TO 3

2 CONTINUE

3 OPEN (3, FILE='JAC.DAT', FORM='FORMATTED', STATUS='UNKNOWN')

WRITE (3,\*) B

CLOSE (3)

END

1. **Выходная программа**

program jac

parameter (l = 8,itmax = 20)

real a(l,l),eps,maxeps,b(l,l),c(l,l)

! arrays A and B with block distribution

print \*, '\*\*\*\*\*\*\*\*\*\* TEST\_JACOBI \*\*\*\*\*\*\*\*\*\*'

maxeps = 5.000000e-008

!nest of two parallel loops, iteration (i,j) will be executed on

!processor, which is owner of element A(i,j)

!$OMP PARALLEL PRIVATE(i)

!$OMP DO

do j = 1,l

do i = 1,l

a(i,j) = 0.

if (i .eq. 1 .or. j .eq. 1 .or. i .eq. l .or. j .eq. l) then

b(i,j) = 0.

else

b(i,j) = 1. + i + j

endif

enddo

enddo

!$OMP END DO

!$OMP END PARALLEL

do it = 1,itmax

eps = 0

!variable EPS is used for calculation of maximum value

!$OMP PARALLEL PRIVATE(i)

!$OMP DO

do j = 2,l - 1

do i = 2,l - 1

a(i,j) = b(i,j)

enddo

enddo

!Copying shadow elements of array A from

!neighbouring processors before loop execution

!$OMP END DO NOWAIT

!$OMP DO

do j = 2,l - 1

do i = 2,l - 1

a(i,j) = (c(i - 1,j) + c(i,j - 1) + c(i + 1,j) + c(i,j +

&1)) / 4

enddo

enddo

!$OMP END DO

!$OMP END PARALLEL

print 200, it,0

200 FORMAT(' IT = ',I4, ' EPS = ', E14.7)

if (0 .lt. 5.000000e-008) goto 3

enddo

3 open (unit = 3,file = 'JAC.DAT',form = 'FORMATTED',status = 'UNKNO

&WN')

write (unit = 3,fmt = \*) b

close (unit = 3)

end

**Общее время последовательного выполнения: 7667 у.е.**

**Общее время параллельного выполнения (4 нити): 2471 у.е.**

**Суммарное ускорение: в 3,1 раза**

# Заключение

В рамках дипломной работы был реализован "Эксперт по распараллеливанию на мультипроцессор", как компонент "Системы автоматизации распараллеливания".

Работа системы была протестирована на примерах, описанных в пункте 5.5. В качестве выходного результата формируется файл с выходной программой, а также файл, описывающий пошаговые действия "Эксперта" с аргументированием причины формирования выходного файла, а также подсчитывающего ускорение программы.

Проект реализован в программе Visual Studio .NET на языке C++. Общий объем программного кода, реализующего "Эксперт"- свыше 1800 строк.

# Перечень принятых терминов

**Потенциально Параллельный Цикл (ППЦ) –** цикл, который можно распараллелить средствами OpenMP, не обязательно эффективно.

**Цикл Неподдающийся Распараллеливанию (ЦНР) –** циклы с зависимостями следующих видов:

1. содержащие ввод/вывод,
2. содержащие вызовы процедур,
3. содержащие выход из цикла,
4. содержащие неизвестные анализатору зависимости,
5. циклы, с которыми не справился анализатор.

**Параллельный регион –** последовательность циклов, линейных фрагментов, ветвлений, которая будет заключена в рамки **!$OMP PARALLEL - !$OMP END PARALLEL**.

**Вариант параллелизма программы (Вариант параллелизма) –** вариантдиректив OpenMP, описывающих параллельные области и параллельные циклы (без директив локализации переменных).

**Вариант локализации переменных (Вариант локализации) –** вариантдиректив локализации переменных. Для каждого варианта параллелизма может существовать несколько вариантов локализации.

**Схема распараллеливания –** фиксация одного варианта параллелизма и одного варианта локализации.

**1-е внутреннее представление –** дерево, по структуре схожее с деревом из Базы Данных, в котором помечены все ППЦ и для каждого ППЦ обозначены пометки для локализации. Является результатом работы 1-го шага.

**2-е внутреннее представление -** дерево, по структуре схожее с деревом из Базы Данных, в котором расставлены пометки, по которым в базу данных можно занести комментарии по распараллеливанию (Распараллеливание и Локализацию). Является результатом работы 3-го шага.

**Наилучшие комментарии –** комментарии такие, что оценочная функция для программы с такими комментариями будет минимальна по сравнению с программой с любыми другими комментариями. Нахождение таких комментариев - главная задача эксперта.

**Проверка Альтернативного Распараллеливания –** нахождения минимума оценочной функции между текущим 2-м внутренним представлением и неким другим (альтернативным) распараллеливанием, с фиксацией распараллеливания с минимальной оценочной функцией во 2-м внутреннем представлении.

**Проверка Альтернативной Локализации –** нахождения минимума оценочной функции для набора локализаций для данного распараллеливания с фиксированием комментариев, соответствующих минимальной оценочной функцией во 2-м внутреннем представлении.