Московский государственный университет им. М.В. Ломоносова

Введение в методы параллельных вычислений

Разработчик:
А.В. Старченко, д.ф.-м.н., профессор
E-mail: starch@math.tsu.ru
Томский государственный университет

Направление 010300.68
«Фундаментальная информатика и информационные технологии»

Проект комиссии Президента по модернизации и техническому развитию экономики России «Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения»

Разработка курса выполнена в рамках Проекта комиссии Президента РФ по модернизации и техническому развитию экономики России «Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения»

Применение потенциала суперкомпьютерных технологий (СКТ) как значимой составляющей инновационного развития страны является задачей государственной важности, относится к приоритетному направлению и находится под постоянным контролем Президента и Правительства России. Одним из сдерживающих факторов развития страны в этом направлении является острая нехватка высококвалифицированных кадров в области СКТ, поскольку подготовка таких специалистов сейчас отсутствует как элемент системы высшего профессионального образования.

Стратегической целью проекта является создание национальной системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения.

http://hpc-education.ru.

Содержание курса

- Введение
- Рекуррентные формулы
- Параллельные вычисления определенных и кратных интегралов
- Умножение матрицы на вектор. Умножение матриц
- Прямые методы решения систем линейных уравнений на многопроцессорных системах Организация межпроцессорных обменов
- Трехдиагональные системы. Параллельная реализация прямых методов решения систем линейных уравнений
- Параллельная реализация итерационных методов решения СЛАУ
- Параллельная реализация быстрого преобразования Фурье

Содержание лекции

- Вычисление скалярного произведения векторов на ***p***-процессорной системе
- Вычисление произведения матрицы на вектор на ***p***-процессорной системе
  - Общая схема построения параллельного алгоритма
  - Достижение максимально возможного быстродействия (***p=n²***)
  - Использование параллелизма среднего уровня (***n<p<n²***)
  - Конвейерная схема
  - Использование ограниченного набора процессорных элементов (***p<<n***)
- Умножение матриц
  - Общая схема построения параллельного алгоритма
  - Блочные алгоритмы умножения матриц: простой блочный алгоритм, алгоритм Кэннона, алгоритм Фокса
- Заключение

Вычисление скалярного произведения векторов

- Рассмотрим базовые алгоритмы линейной алгебры: вычисление скалярного произведения, матрично-векторное умножение и умножение матриц
- Пусть $x, y \in \mathbb{R}^n$ и требуется найти значение скалярного произведения векторов

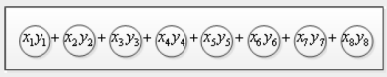
$$(x, y) = \sum_{i=1}^{n} x_i y_i$$

- Алгоритм требует ***n*** умножений и ***n-1*** сложений. Для простоты выкладок время выполнения последовательного алгоритма можно оценить как

$$T_1 = (t_{mult} + t_{add}) \cdot n$$

Вычисление скалярного произведения

- Следуя общей методологии построения алгоритмов для параллельных вычислений:
  - Декомпозиция. Разобьем задачу на мелкозернистые подзадачи – ***i***-ая подзадача запоминает x_i и y_i и вычисляет произведение $x_i \cdot y_i$
  - Проектирование коммуникаций. Коммуникации между подзадачами («+») устанавливаются таким образом, чтобы обеспечить суммирование полученных произведений.



Вычисление скалярного произведения

- Укрупнение подзадач производится путем объединения ***n/p*** (***p*** - предполагаемое число процессорных элементов) блоков мелкозернистых подзадач с сохранением коммуникаций между укрупненными блоками
- Планирование вычислений. Каждый блок назначается на выполнение одному процессорному элементу

Время на вычисление скалярного произведения без учета межпроцессорной передачи данных можно оценить как

$$T_p \approx (t_{mult} + t_{add}) \cdot \frac{n}{p} + t_{add} \cdot \log_2 p$$

Умножение матрицы на вектор

- При умножении квадратной матрицы $A \in \mathbb{R}^{n \times n}$ на вектор $x \in \mathbb{R}^n$ компоненты вектора-результата вычисляются по формуле

$$y_i = \sum_{j=1}^{n} a_{ij} \cdot x_j; 1 \le i \le n; \quad T_1 \approx (t_{mult} + t_{add}) \cdot n^2$$

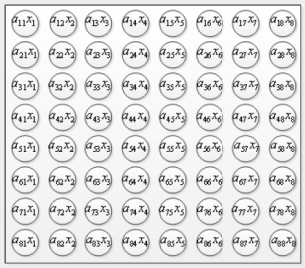
- При выборе подходов распараллеливания этих формул можно заметить:
  - Вычисление каждого y_i осуществляется независимо и одновременно и может быть выполнено параллельно
  - Умножение каждой строки на вектор включает независимые операции поэлементного умножения, которые могут быть выполнены параллельно
  - Суммирование может осуществляться по алгоритму сдваивания

Построение алгоритма для параллельных вычислений

Декомпозиция.

Задачу разбиваем на мелкие задачи, каждая суть вычисление произведения $a_{ij} \cdot x_j; i, j = 1, \ldots, n.$

Всего таких подзадач будет n^2. Каждая подзадача ***(i,j)*** должна иметь информацию о коэффициенте матрицы a_{ij} и компоненте вектора x_j

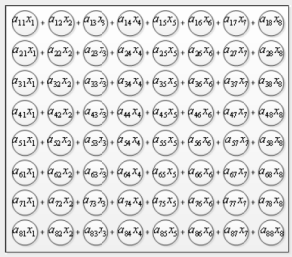


Построение алгоритма для параллельных вычислений

Проектирование коммуникаций.

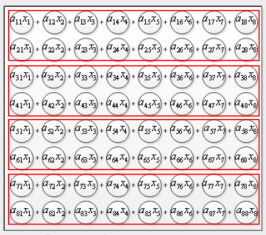
Для получения значения компоненты вектора y с использованием каскадной схемы суммирования необходимо организовать коммуникации между подзадачами

$(i,1),(i,2),\ldots,(i,n)$

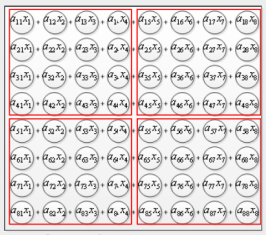


Построение алгоритма для параллельных вычислений

Укрупнение. Поводится на основе одномерной или двумерной стратегии объединения мелкозернистых подзадач.



n = 8 p = 4



n = 8 p = 4

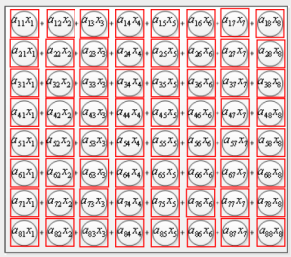
Достижение максимально возможного быстродействия (***p=n²***)

- В этом случае алгоритм можно разбить на подзадачи, в каждой из которых запоминаются значения элемента матрицы a_{ij} и компоненты вектора x_j и рассчитывается их произведение.
- Коммуникации между подзадачами проектируются таким образом, чтобы организовать по каскадной схеме суммирование соответствующих произведений для вычисления отдельной компоненты вектора y_i .
- Оценим время и ускорение:

$$S_p = \frac{T_1}{T_p} \approx \frac{(t_{mult} + t_{add}) \cdot n^2}{t_{mult} + t_{add} \cdot \log_2 n} \approx \frac{4p}{2 + \log_2 p}$$

Достижение максимально возможного быстродействия (*p=n²*)

- Для этого случая наиболее подходящими будут топологии, в которых обеспечивается быстрая передача данных в каскадной схеме суммирования: полный граф или гиперкуб.
- Другие топологии приводят к возрастанию коммуникационных затрат из-за удлинения маршрутов передачи данных.



Достижение максимально возможного быстродействия (*p=n²*)

- Умножение матрицы на вектор на систолическом массиве процессоров (***p=n²***).
- В таком массиве процессорные элементы находятся в узлах регулярной решетки, в которой роль ребер играют межпроцессорные элементы.
- Все процессоры управляются общим тактовым генератором. В каждом цикле работы любой процессорный элемент (ПЭ) получает данные от соседних ПЭ, выполняет одну команду и передает данные соседям.

Достижение максимально возможного быстродействия (*p=n²*)

- Систолический алгоритм:
- Элементы матрицы ***A*** распределяют по ПЭ так, что элементы ***j***-ой строки матрицы сдвигаются циклически влево на ***j-1*** позицию (***n-1*** одиночный сдвиг).

$$A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{22} & a_{23} & \dots & \dots & a_{21} \\ a_{33} & a_{34} & \dots & \dots & a_{32} \\ \dots & \dots & \dots & \dots & \dots \\ a_{nn} & a_{n1} & \dots & \dots & a_{n\,n-1} \end{pmatrix}$$

Достижение максимально возможного быстродействия (*p=n²*)

- На основе вектора ***x*** создается матрица ***X*** такая, что каждый последующий столбец, начиная со второго, получается циклическим сдвигом предыдущего столбца на одну позицию вверх.

$$X = \begin{pmatrix} x_1 & x_2 & \dots & \dots & x_n \\ x_2 & x_3 & \dots & \dots & x_1 \\ x_3 & x_4 & \dots & \dots & x_2 \\ \dots & \dots & \dots & \dots & \dots \\ x_n & x_1 & \dots & \dots & x_{n-1} \end{pmatrix}$$

Достижение максимально возможного быстродействия (*p=n²*)

- Инициализация:

a_{11}, x_1, y_1 a_{12}, x_2, y_1 a_{13}, x_3, y_1

a_{22}, x_2, y_2 a_{23}, x_3, y_2 a_{21}, x_1, y_2

a_{33}, x_3, y_3 a_{31}, x_1, y_3 a_{32}, x_2, y_3

- Каждый процессор производит умножение пары чисел и записывает результат в y_i, передает a_{ij} левому соседу в решетке, а x_j – верхнему. На следующем цикле данные принимаются от соседних процессорных элементов, снова выполняется умножение пары чисел, суммирование его с y_i и передача данных по ребрам решетки.

Достижение максимально возможного быстродействия (*p=n²*)

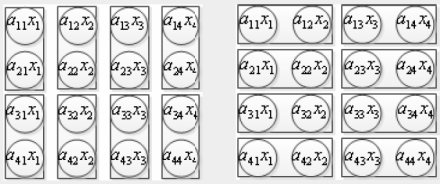
- После окончания алгоритма на каждом ПЭ получается компонента вектора ***y***. В данном алгоритме не используется каскадная схема суммирования.
- Сравнивая показатели эффективности рассмотренных алгоритмов для ***p=n²*** можно отметить более низкую производительность алгоритма на систолическом массиве процессоров.

$$S_p = \frac{T_1}{T_p} \approx \frac{(t_{mult} + t_{add})\,n^2}{(t_{mult} + t_{add})\,n} = n = \sqrt{p};$$

- Наиболее подходящей топологией для вычисления произведения матрицы на вектор на систолическом массиве процессоров является тор или полный граф.

Использование параллелизма среднего уровня ($n<p<n^2$)

- Пусть $p=k \cdot n$ ($1<k<n$). В этом случае декомпозиция задачи может быть осуществлена двумя способами:
  - либо с последующим применением обычной каскадной схемы суммирования,
  - либо с использованием модифицированной каскадной схемы.



Использование параллелизма среднего уровня ($n<p<n^2$)

- Сравним две схемы декомпозиции (1) и (2), получим:

$$T_p^{(1)} \approx t_{mult}\frac{n^2}{p} + t_{add}\frac{n^2}{p}\log_2 n;$$

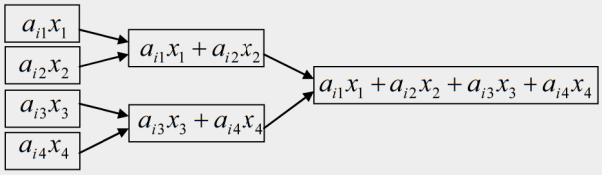
$$T_p^{(2)} \approx (t_{add} + t_{mult})\frac{n^2}{p} + t_{add}\log_2\left(\frac{p}{n}\right);$$

- Второй вариант более предпочтителен, например:
- при $k = n/2 : T_p^{(1)} \approx 2t_{mult} + 2t_{add}\log_2 n;$

$$T_p^{(2)} \approx 2t_{mult} + t_{add}(1 + \log_2 n);$$

Использование параллелизма среднего уровня ($n<p<n^2$)

- При $p=2 \cdot n$ ($k=2$) можно организовать конвейерное умножение матрицы на вектор
- Множество всех $2 \cdot n$ процессорных элементов разбивается на непересекающиеся группы $Q_0, Q_1, \ldots, Q_{log2n}$, в каждой из которых $n/2^i$ ($i=0,\ldots,\log_2 n$) ПЭ.



Использование параллелизма среднего уровня ($n<p<n^2$)

- Группа Q_0, объединяющая n процессорных элементов, используется для реализации поэлементного умножения i-ой строки матрицы A на вектор x.
- На каждом такте работы конвейера следующая строка матрицы поступает на процессоры группы Q_0. Следует отметить, что в один момент времени конвейер обрабатывает $\log_2 n+1$ строк матрицы A.
- При конвейерной реализации умножения матрицы на вектор временные затраты выше.
- Преимущества: меньшее количество пересылок данных между процессорами и возможность получения части результатов до окончания всего вычислительного процесса.
- Топология – двоичное дерево.

Использование ограниченного набора процессоров ($p<<n$)

- На каждой из имеющихся процессорных элементов пересылается вектор x и n/p ($n \bmod p=0$) строк матрицы A.
- Выполнение операции умножения строк матрицы на вектор выполняется при помощи обычного последовательного алгоритма.
- С математической точки зрения этот алгоритм эквивалентен умножению в блочной форме:

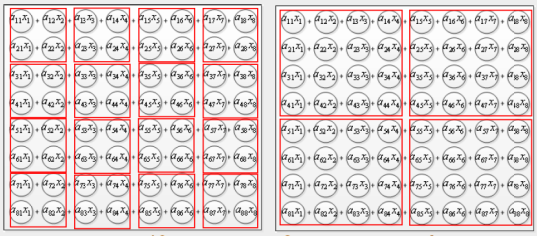
$$y = Ax = \begin{bmatrix} A_1 \\ \ldots \\ A_p \end{bmatrix} x = \begin{bmatrix} A_1 x \\ \ldots \\ A_p x \end{bmatrix}; \quad T_p \approx (t_{mult} + t_{add})\frac{n^2}{p}.$$

Использование ограниченного набора процессоров ($p<<n$)

- При принятом распределении данных по процессорам вычисления произведений осуществляются с максимальным параллелизмом.
- Аналогичные соображения применимы к разбиению матрицы A на группы столбцов.
- Необходимо следить за обеспечением равномерной загрузки процессорных элементов при $n \bmod p \neq 0$.
- Топология – звезда: управляющий ПЭ обеспечивает загрузку вычислительных узлов исходными данными и собирает результаты проведенных вычислений.

Использование ограниченного набора процессоров (*p<<n*)

- Рассмотрим двумерное укрупнение мелкозернистых подзадач (***p=s²*** и ***n mod s=0***). На каждом ПЭ размещается ***n²/p*** элементов матрицы ***A*** и ***n/s*** компонентов вектора ***x***.



p=16 ***n=8*** ***p=4***

Использование ограниченного набора процессоров (*p<<n*)

- Каждый ПЭ локально выполняет ***n²/p*** умножений и ***n²/p*** сложений полученных произведений.
- Затем по алгоритму сдваивания осуществляется суммирование ***n/s*** результатов с ПЭ, расположенных в одном ряду, для определения ***n/s*** компонентов вектора ***y***.

$$T_p \approx \frac{n^2}{p} t_{muli} + \frac{n^2}{p} t_{add} + t_{add} \frac{n}{\sqrt{p}} \log_2 \sqrt{p}$$

- При двумерном укрупнении ускорение параллельного алгоритма несколько ниже, чем при одномерном, что обусловлено использованием алгоритма сдваивания.
- Затраты на инициализацию обоих алгоритмов примерно одинаковы.

Умножение матриц

- Задача матричного умножения требует для своего решения выполнения большого количества арифметических операций.
- Пусть ***A, B, C*** – квадратные матрицы ***nxn***, ***C=AB***. Тогда компоненты матрицы ***C*** рассчитываются по следующей формуле $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}; i, j = 1,...,n.$
- Видно, что для вычисления одного элемента матрицы ***C*** требуется ***n*** умножений и ***n*** сложений. Тогда

$$T_1 \approx (t_{mult} + t_{add}) n^3.$$

Умножение матриц

- Произведение матриц может рассматриваться как ***n²*** независимых скалярных произведений, либо как ***n*** независимых произведений матрицы на вектор. В каждом случае используются различные алгоритмы.
- Производительность умножения матриц может быть улучшена путем изменения порядка циклов ***i-j-k***.
- Повышение быстродействия умножения матриц за счет эффективного использования кэш-памяти можно обеспечить с использованием блочного представления матриц.
- Процедура параллельного умножения матриц легко может быть построена на основе полученных выше алгоритмов скалярного умножения векторов или матрично-векторного умножения для многопроцессорных систем с распределенной памятью.

Умножение матриц (общий подход)

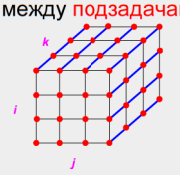
- *Декомпозиция.* На основе проведенного выше анализа выберем в качестве части проблемы (подзадачи) вычисление произведения $a_{ik} \cdot b_{kj}$. Количество таких подзадач равно ***n³***, и их можно представить в виде трехмерного массива. Данные, необходимые для вычисления произведения, определяются компонентами матриц a_{ik} и b_{kj}.
- *Проектирование коммуникаций.* Для вычисления значения элемента матрицы c_{ij} требуется обеспечение коммуникаций между подзадачами, в которых производится расчет произведений $a_{ik} \cdot b_{kj}$, (***k=1,...,n***), для последующего суммирования произведений.

Умножение матриц (общий подход)

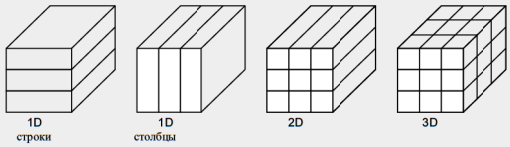
- *Укрупнение.* Для имеющихся мелкозернистых подзадач могут быть выбраны следующие естественные стратегии их объединения в ***p*** (***p<<n³***) подзадач:
  - одномерное укрупнение по строкам.
  - одномерное укрупнение по столбцам.
  - двумерное укрупнение. Здесь реализуется алгоритм, основанный на блочном представлении матриц.
  - трехмерное укрупнение.
- *Планирование вычислений.* На этом этапе разработки параллельного алгоритма необходимо определить, где, на каких процессорных элементах будут выполняться укрупненные подзадачи.

Умножение матриц (общий подход)

- Коммуникации между подзадачами



- Стратегии укрупнения мелкозернистых подзадач



Умножение матриц

- Умножение матриц фиксированного размера за меньшее время можно обработать с использованием 3D-декомпозиции, поскольку для вычислений привлекается большее количество процессорных элементов.

$$T_p \approx (t_{mult} + t_{add}) \cdot n^3 / p;$$

$$S_p \approx \frac{(t_{mult} + t_{add}) \cdot n^3}{(t_{mult} + t_{add}) \cdot n^3 / p} = p;$$

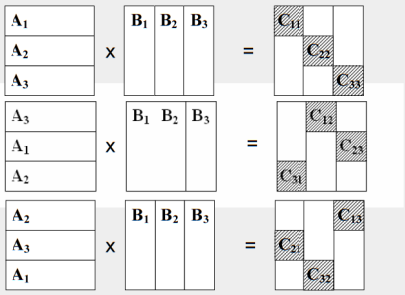
- Алгоритмы, рассмотренные выше, требуют значительных ресурсов памяти многопроцессорной вычислительной системы для хранения блоков матриц.

Простой блочный алгоритм

- Матрица ***A*** распределяется по процессорам блоками, содержащими ***(n/p)xn*** элементов, ***B*** – ***nx(n/p)*** элементов.
- Проводится матричное умножение соответствующих блоков, и в результате определяются блоки матрицы ***C*** размером ***(n/p)x(n/p)***, стоящие на главной диагонали.
- На каждой следующей итерации производится обмен блоками матрицы ***A*** между ПЭ циклическим сдвигом по уменьшению номера процессора (или увеличению), вычисления произведения блочных матриц.
- После ***p*** итераций расчет матрицы ***C*** заканчивается.
- Такой способ параллельного умножения матриц может быть использован при одномерном укрупнении подзадач.

Простой блочный алгоритм

- Этап 1
- Этап 2
- Этап 3



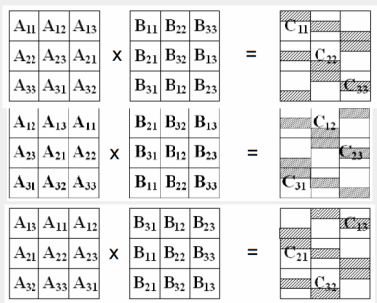
- Объем памяти для 1ПЭ – ***3n²/p*** чисел; суммарный объем пересылаемых данных – ***n²*** чисел

Алгоритм Кэннона

- Имеется процессорная 2D решетка из ***p=s²*** процессорных элементов ПЭ***ij***, ***i,j=1,…,s***, в которой первый и последний узлы как в ряду процессорной решетки, так и в ее столбце имеют прямое соединение.
- Матрицы ***A*** и ***B*** делятся на ***p=s²*** блоков согласованных размеров – ***Aij,Bij; i,j=1,…,s***. На каждый ПЭ***ij*** рассылаются соответствующие блоки ***Ai,j+i-1***, ***Bi+j-1,j***. ПЭ***ij*** будет рассчитывать ***Cij***.
- Затем на каждой итерации:
  - Вычисляется произведение блоков.
  - ПЭ***ij*** отправляет имеющийся блок ***A*** на ПЭ***i,j-1***, ***B*** - на ПЭ***i-1,j.***
- После выполнения ***s*** итераций на каждом ПЭ***ij*** получается блок ***Cij***.

Алгоритм Кэннона

- Этап 1
- Этап 2
- Этап 3



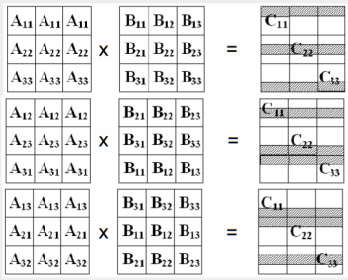
- Объем памяти для 1ПЭ – ***3n²/p*** чисел; суммарный объем пересылаемых данных – ***2n²/s*** чисел

Алгоритм Фокса

- Имеется процессорная 2D решетка из $p=s^2$ процессорных элементов ПЭij, $i,j=1,\ldots,s$, в которой первый и последний узлы как в ряду процессорной решетки, так и в ее столбце имеют прямое соединение.
- Матрицы A и B делятся на $p=s^2$ блоков согласованных размеров – Aij,Bij; $i,j=1,\ldots,s$. На каждый ПЭij рассылаются соответствующие блоки Aij, Bij. ПЭij будет рассчитывать Cij.
- Затем на каждой итерации $l=0,\ldots,s-1$:
  - Для каждого i вычисляем $m=(i+l-1) \bmod s +1$ и блок Aim пересылается на все ПЭ ряда i.
  - Вычисляется произведение блоков, а результат добавляется к блоку Cij.
  - ПЭij отправляет имеющийся блок B на ПЭ$i-1,j$.
- После выполнения s итераций на каждом ПЭij получается блок Cij.

Алгоритм Фокса

- Этап 1
- Этап 2
- Этап 3



- Объем памяти для 1ПЭ – $4n^2/p$ чисел, суммарный объем пересылаемых данных – $2n^2/s$ чисел

Блочные алгоритмы умножения

- Блочное представление матриц в рассмотренных выше алгоритмах умножения матриц приводит к некоторому повышению объёма пересылаемых между процессорными элементами данных.
- Объем пересылаемых данных может быть уменьшен, если использовать строковое для A и столбцовое для B блочное распределение данных по процессорным элементам.
- Блочное представление матриц при параллельном вычислении матричных произведений имеет ряд преимуществ:
  - пересылка данных является распределённой во времени, что позволяет совмещать вычисления и передачу данных;
  - блочная структура может быть использована при решении слабо заполненных СЛАУ.

Заключение

- Рассмотрены базовые алгоритмы линейной алгебры: вычисление скалярного произведения векторов, произведения матрицы на вектор, умножения матриц и способы их распараллеливания на p-процессорной системе с распределенной памятью.
- Для параллельных алгоритмов умножения матрицы на вектор проведено исследование выбора оптимальной топологии p-процессорной системы от числа имеющихся процессорных элементов.
- Для алгоритмов умножения матриц наряду со способами повышения быстродействия вычислительного процесса рассмотрены блочные алгоритмы, позволяющие за счет использования параллелизма увеличить размер решаемой задачи.

Вопросы для обсуждения

- Опишите как с использованием общей схемы построения параллельных алгоритмов получить параллельную программу скалярного умножения векторов.
- Перечислите возможные подходы распараллеливания матрично-векторного умножения.
- Дайте сравнительный анализ достоинств и недостатков одномерного и двумерного укрупнения при построении параллельного алгоритма умножения матрицы на вектор.
- Опишите параллельный алгоритм матрично-векторного умножения при максимально необходимом для решения этой задачи числа процессоров.
- Чем объяснить преимущество топологии «полный граф» по сравнению с топологией «линейка» для случая $p=n^2$?
- Какой алгоритм суммирования используется при умножении матриц в систолическом массиве процессоров и какая топология является наиболее подходящей в этом случае?
- Опишите как работает конвейерная схема умножения матрицы на вектор при $p=2n$.
- Оцените время работы параллельного алгоритма конвейерной схемы умножения матрицы на вектор при $p=2n$.

Вопросы для обсуждения

- В чем заключается преимущество конвейерного способа решения задачи?
- Какие способы декомпозиции задачи матрично-векторного умножения возможны при ограниченном числе используемых процессоров? Дайте сравнительный анализ.
- Какая топологическая структура наиболее подходит для параллельного матрично-векторного умножения возможны при ограниченном числе используемых процессоров?
- Как обеспечить баланс загрузки процессоров при $n \bmod p \neq 0$?
- Какие способы повышения производительности умножения матриц известны?
- Какие способы декомпозиции задачи матричного умножения возможны?
- Можно ли рассматривать параллельные вычисления не с точки зрения быстродействия получения результата, а как средство для решения сверхбольших задач, которые не помещаются в память обычного компьютера?
- Какие блочные алгоритмы умножения матриц известны? Преимущества и недостатки.
- Как можно сократить затраты на пересылку данных в алгоритме Фокса?

Темы заданий для самостоятельной работы

- Организация параллельных вычислений произведения матрицы на вектор при $p=n$.
- Написать MPI-программу умножения матрицы на вектор для $p=n^2$.
- Написать MPI-программу умножения матрицы на вектор для конвейерной схемы.
- Написать MPI-программу умножения матриц с помощью блочного алгоритма.

Основная литература

1. *Гергель В.П.* Теория и практика параллельных вычислений. – М.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2007.
2. *Гергель В.П., Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем. – Нижний Новгород: Изд-во ННГУ, 2001.
3. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем. – М.:Мир, 1991.
4. *Деммель Дж.* Вычислительная линейная алгебра. – М.:Мир, 2001.
5. *Высокопроизводительные вычисления на кластерах*. – Томск: Изд-во Том. ун-та, 2008.

Следующая тема

- Введение
- Рекуррентные формулы
- Параллельные вычисления определенных и кратных интегралов
- Умножение матрицы на вектор. Умножение матриц
- Прямые методы решения систем линейных уравнений на многопроцессорных системах Организация межпроцессорных обменов
- Трехдиагональные системы. Параллельная реализация прямых методов решения систем линейных уравнений
- Параллельная реализация итерационных методов решения СЛАУ
- Параллельная реализация быстрого преобразования Фурье