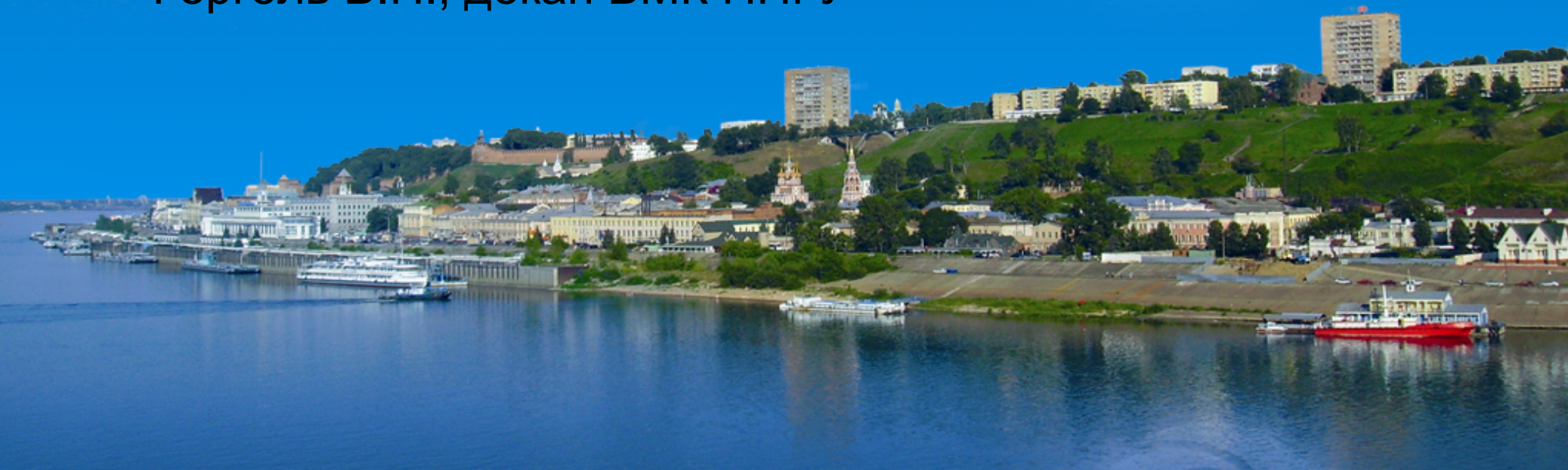
Нижегородский государственный университет
им. Н.И. Лобачевского
- Национальный исследовательский университет -

Лекция.

Моделирование параллельных вычислений

Гергель В.П., декан ВМК ННГУ



- **Суперкомпьютерные технологии** являются базовой критической технологией, поскольку являются сегодня важнейшими во всем спектре технологий, которыми владеет человечество. Именно на их основе решаются наиболее трудные и ресурсоемкие междисциплинарные задачи современной науки, техники, промышленности и бизнеса
- **Суперкомпьютерные технологии** могут явиться сегодня локомотивом развития страны точно также, как в **30-х годах** основой прогресса была авиация, в **40-х годах** – атомное оружие, в **50-60-х годах** – ракетная техника и космос

Содержание

- **Моделирование параллельных вычислений**
  - Показатели эффективности
  - Модель вычислений на основе графа информационных зависимостей
  - Оценки максимально-достижимого параллелизма

Модели параллельных вычислений

- Принципиальный момент при разработке параллельных алгоритмов - **анализ эффективности использования параллелизма**:
  - Оценка эффективности распараллеливания конкретных выбранных методов выполнения вычислений,
  - Оценка максимально возможного ускорения процесса решения рассматриваемой задачи (анализ всех возможных способов выполнения вычислений)

Показатели эффективности параллельного алгоритма…

- **Ускорение (*speedup*)**

  получаемое при использовании параллельного алгоритма для ***p*** процессоров, по сравнению с последовательным вариантом выполнения вычислений, определяется величиной

$$S_p(n) = T_1(n) / T_p(n)$$

  (величина ***n*** используется для параметризации вычислительной сложности решаемой задачи и может пониматься, например, как количество входных данных задачи)

Показатели эффективности параллельного алгоритма…

- **Эффективность (*efficiency*)**

  использования параллельным алгоритмом процессоров при решении задачи определяется соотношением:

$$E_p(n) = T_1(n)/(pT_p(n)) = S_p(n)/p$$

  (величина эффективности определяет среднюю долю времени выполнения параллельного алгоритма, в течение которого процессоры реально используются для решения задачи)

Показатели эффективности параллельного алгоритма…

- **Замечания**
  - Сверхлинейное (*superlinear*) ускорение $S_p(n)>p$ может иметь место в силу следующего ряда причин:
    - неравноправность выполнения последовательной и параллельной программ (например, недостаток оперативной памяти),
    - нелинейный характер зависимости сложности решения задачи от объема обрабатываемых данных,
    - различие вычислительных схем последовательного и параллельного методов.
  - Показатели качества параллельных вычислений являются противоречивыми: попытки повышения качества параллельных вычислений по одному из показателей (ускорению или эффективности) может привести к ухудшению ситуации по другому показателю.

Показатели эффективности параллельного алгоритма

- **Стоимость (*cost*)** вычислений

$$C_p = pT_p$$

- **Стоимостно-оптимальный** (***cost-optimal***) параллельный алгоритм - метод, стоимость которого является пропорциональной времени выполнения наилучшего последовательного алгоритма.

Оценка максимально достижимого параллелизма…

- Оценка качества параллельных вычислений предполагает знание *наилучших* (*максимально достижимых*) значений показателей ускорения и эффективности
- Получение идеальных величин $S_p=p$ для ускорения и $E_p=1$ для эффективности может быть обеспечено не для всех вычислительно трудоемких задач

Оценка максимально достижимого параллелизма

❑ Закон Амдаля (*Amdahl*)

Достижению максимального ускорения может препятствовать существование в выполняемых вычислениях последовательных расчетов, которые не могут быть распараллелены.

Пусть ***f*** есть *доля последовательных вычислений* в применяемом алгоритме обработки данных.

Ускорение процесса вычислений при использовании ***p*** процессоров ограничивается величиной:

$$S_p \le \frac{1}{f + (1-f)/p} \le S^* = \frac{1}{f}$$

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Граф информационных зависимостей…

- Модель в виде графа "операции-операнды" используется для описания существующих информационных зависимостей в выбираемых алгоритмах
- В наиболее простом виде модель основывается на предположениях:
  - время выполнения любых вычислительных операций является одинаковым и равняется 1,
  - передача данных между вычислительными устройствами выполняется мгновенно без каких-либо затрат времени.

МОДЕЛИ ВЫЧИСЛЕНИЙ. Граф информационных зависимостей…

Множество операций, выполняемые в исследуемом алгоритме решения вычислительной задачи, и существующие между операциями информационные зависимости могут быть представлены в виде *ациклического ориентированного графа*

$$G = (V, R)$$

$V = \{1,...,|V|\}$ – множество вершин графа, представляющих выполняемые операции алгоритма,

R – множество дуг графа; дуга *r(i,j)* принадлежит графу только если операция *j* использует результат выполнения операции *i*

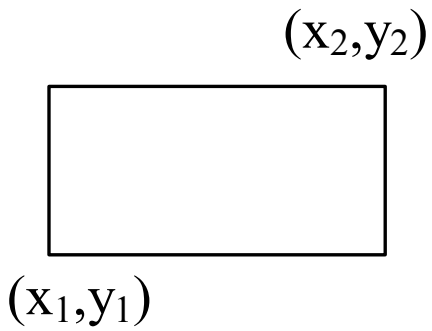
Вершины без входных дуг могут использоваться для задания операций ввода, а вершины без выходных дуг – для операций вывода.

$\overline{V}$ – множество вершин графа без вершин ввода,

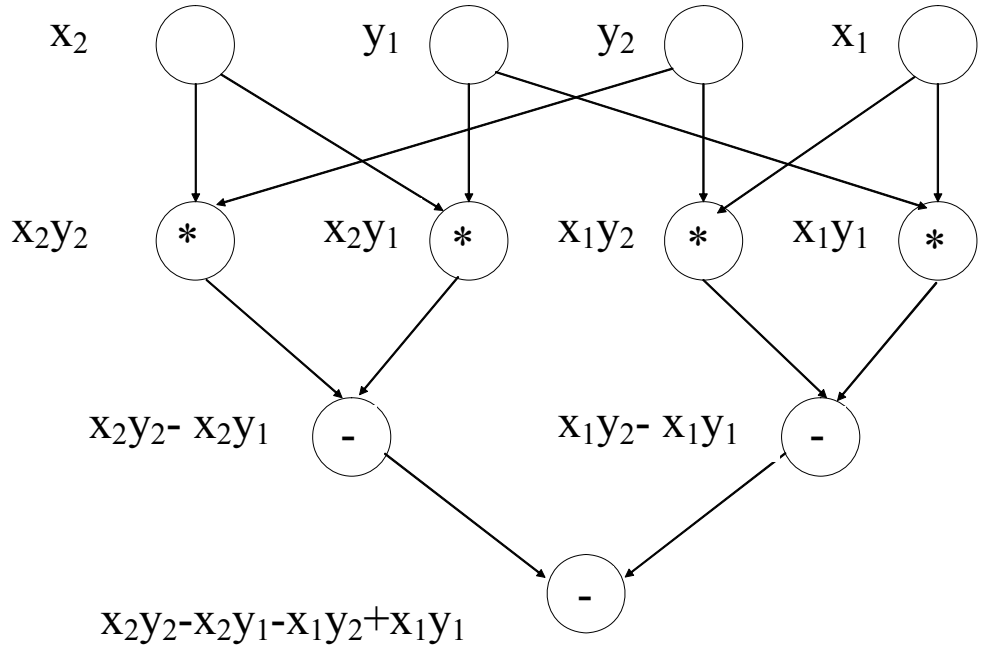
$d(G)$ – диаметр графа (длина максимального пути)

МОДЕЛИ ВЫЧИСЛЕНИЙ. Граф информационных зависимостей…

Пример: граф алгоритма вычисления площади прямоугольника, заданного координатами двух противолежащих углов



$$S = (x_2-x_1)(y_2-y_1) = x_2y_2-x_2y_1-x_1y_2+x_1y_1$$



МОДЕЛИ ВЫЧИСЛЕНИЙ.
Граф информационных зависимостей

- Операции алгоритма, между которыми нет пути в рамках выбранной схемы вычислений, могут быть выполнены **параллельно**
- Схемы вычислений обладают различными возможностями для распараллеливания, при построении модели вычислений может быть поставлена **задача выбора наиболее подходящей** для параллельного исполнения вычислительной схемы алгоритма
-

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Схема параллельного выполнения алгоритма

- Пусть ***p*** есть количество процессоров, используемых для выполнения алгоритма. Тогда для параллельного выполнения вычислений необходимо задать множество (*расписание*):

$$H_p = \{(i, P_i, t_i) : i \in V\}$$

  - ***i*** - есть номер операции,
  - P_i - есть номер процессора,
  - t_i - есть время начала выполнения *i*-ой операции.
- Должны выполняться условия:
  - один и тот же процессор не должен назначаться разным операциям в один и тот же момент времени:

$$\forall i, j \in V : t_i = t_j \Rightarrow P_i \neq P_j$$

  - к назначаемому моменту выполнения операции все необходимые данные уже должны быть вычислены:

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Определение времени выполнения алгоритма…

- Модель параллельного алгоритма:

$$A_p(G, H_p)$$

- Время выполнения параллельного алгоритма с заданным расписанием:

$$T_p(G, H_p) = \max_{i \in V}(t_i + 1)$$

- Время выполнения параллельного алгоритма с оптимальным расписанием:

$$T_p(G) = \min_{H_p} T_p(G, H_p)$$

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Определение времени выполнения алгоритма…

- Минимально возможное время решения задачи при заданном количестве процессоров (определение *наилучшей вычислительной схемы*):

$$T_p = \min_G T_p(G)$$

- Оценка наиболее быстрого исполнения алгоритма (при использовании *паракомпьютера* – системы с неограниченным числом процессоров):

$$T_\infty = \min_{p \geq 1} T_p$$

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Определение времени выполнения алгоритма…

- Время выполнения последовательного алгоритма для заданной вычислительной схемы:

$$T_1(G) = \left|\overline{V}\right|$$

- Время выполнения последовательного алгоритма:

$$T_1 = \min_G T_1(G)$$

- Время последовательного решения задачи:

$$T_1^* = \min T_1$$

Подобные оценки необходимы для определения эффекта использования параллелизма

- **Теорема 1**

  Минимально возможное время выполнения параллельного алгоритма определяется длиной максимального пути вычислительной схемы алгоритма:

$$T_\infty(G) = d(G)$$

- **Теорема 2**

  Пусть для некоторой вершины вывода в вычислительной схеме алгоритма существует путь из каждой вершины ввода. Кроме того, пусть входная степень вершин схемы (количество входящих дуг) не превышает 2. Тогда минимально возможное время выполнения параллельного алгоритма ограничено снизу значением:

$$T_\infty(G) = \log_2 n,$$

  где ***n*** есть количество вершин ввода в схеме алгоритма

- **Теорема 3**

  При уменьшении числа используемых процессоров время выполнения алгоритма увеличивается пропорционально величине уменьшения количества процессоров, т.е. :

$$\forall q = cp, \quad 0 < c < 1 \Rightarrow T_p \le cT_q$$

- **Теорема 4**

Для любого количества используемых процессоров справедлива следующая верхняя оценка для времени выполнения параллельного алгоритма:

$$\forall p \Rightarrow T_p < T_\infty + T_1 / p$$

Теорема 5

Времени выполнения алгоритма, которое сопоставимо с минимально возможным временем T_∞, можно достичь при количестве процессоров порядка $p \sim T_1/T_\infty$, а именно:

$$p \geq T_1 / T_\infty \Rightarrow T_p \leq 2T_\infty$$

При меньшем количестве процессоров время выполнения алгоритма не может превышать более, чем в 2 раза, наилучшее время вычислений при имеющемся числе процессоров, т.е.:

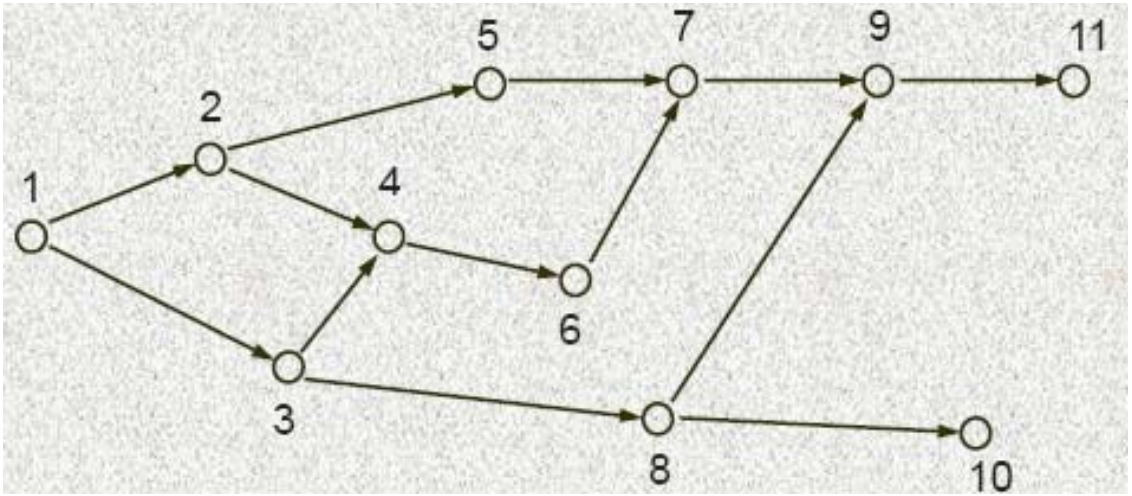
$$p < T_1 / T_\infty \Rightarrow \frac{T_1}{p} \leq T_p \leq 2\frac{T_1}{p}$$

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Определение времени выполнения алгоритма

- **Рекомендации**
  - при выборе вычислительной схемы алгоритма должен использоваться граф с минимально возможным диаметром (теорема 1),
  - для параллельного выполнения целесообразное количество процессоров определяется величиной $p \sim T_1/T_\infty$ (теорема 5),
  - время выполнения параллельного алгоритма ограничивается сверху величинами, приведенными в теоремах 4 и 5.

Строгая параллельная форма

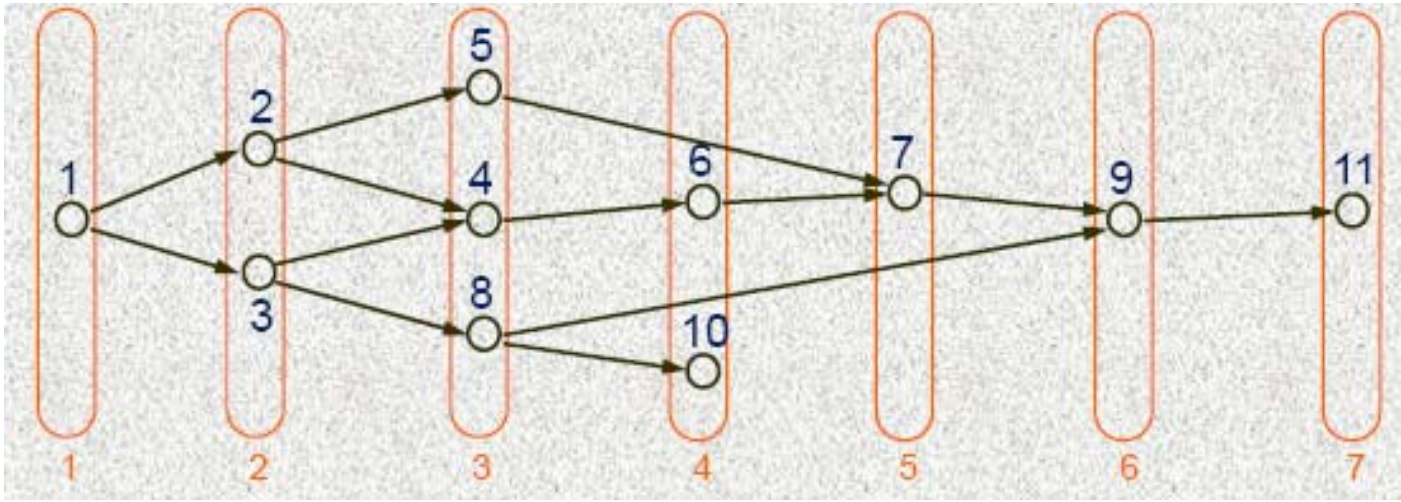
- Вершины графа помечаются одним из индексов 1,2,…,*s*, *s*<*n*,
- Разметка должна быть такой, что, если из вершина *i* есть дуга в вершину *j*, то *i*<*j*



- Группа вершин с одинаковыми индексами называется *ярусом*

МОДЕЛИ ВЫЧИСЛЕНИЙ. Параллельные формы алгоритма…

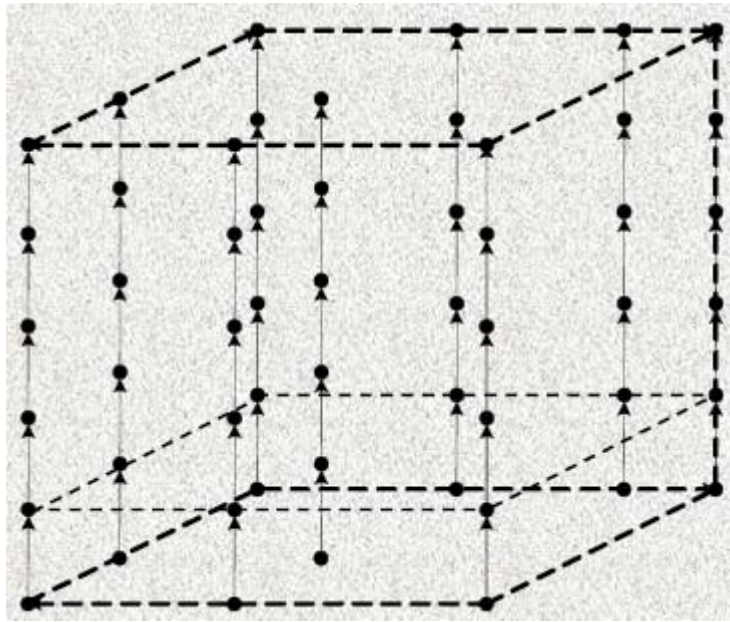
- **Каноническая параллельная форма**
  - Максимальная длина пути для вершины *i* равна *i*-1



- **Такая форма существует и она единственна**

- **Пример: Граф операции матричного умножения**

```
for (i=0; i<n; i++ )
  for (j=0; j<n; j++ ) {
    a[i][j] = 0;
    for (k=0; k<n; k++ )
      a[i][j] = a[i][j] +
                b[i][k]*c[k][j];
}
```



Пример: Вычисление частных сумм…

- Задача нахождения частных сумм последовательности числовых значений (*prefix sum problem*):

$$S_k = \sum_{i=1}^{k} x_i, 1 \le k \le n$$

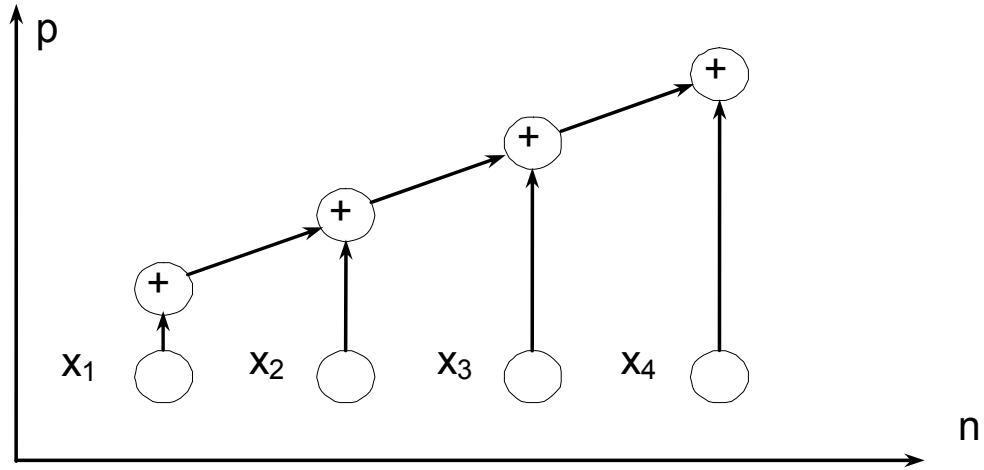
- Задача вычисления общей суммы имеющегося набора значений:

$$S = \sum_{i=1}^{n} x_i$$

Пример: Вычисление частных сумм…

- Последовательный алгоритм суммирования элементов числового вектора

$$S = \sum_{i=1}^{n} x_i, \qquad G_1 = (V_1, R_1)$$

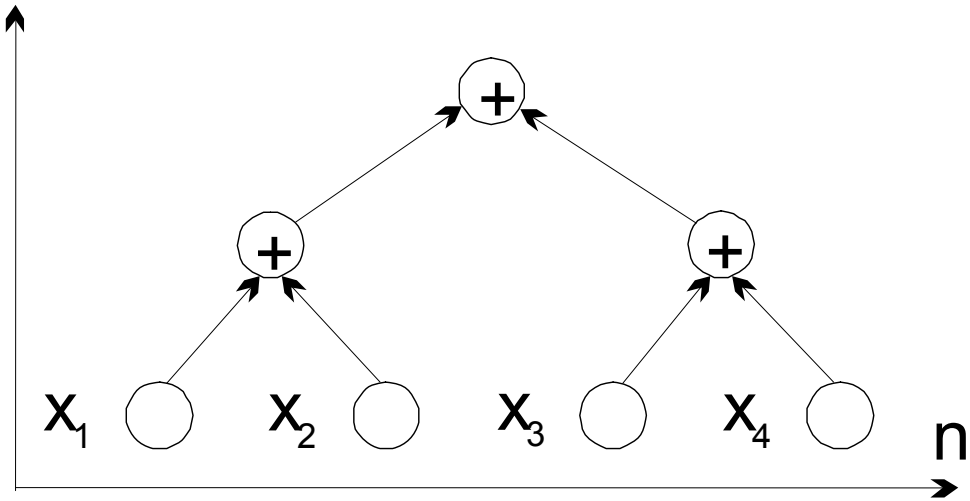


Данный "стандартный" алгоритм суммирования допускает только ***строго последовательное исполнение*** *и не может быть распараллелен*

Пример: Вычисление частных сумм…

- **Каскадная схема суммирования**

$$G_2 = (V_2, R_2)$$



- V_2 = { ($v_{i1},\ldots,v_{ili}$), $0 \le i \le k$, $1 \le l_i \le 2^{-i}n$ } есть вершины графа,
- ($v_{01},\ldots, v_{0n}$) есть операции ввода,
- ($v_{11},\ldots,v_{1n/2}$) есть операции первой итерации и т.д.,
- R_2 = { ($v_{i-1,2j-1}v_{ij}$),($v_{i-1,2j}v_{ij}$), $1 \le i \le k$, $1 \le l_i \le 2^{-i}n$ } есть множество дуг графа.

Пример: Вычисление частных сумм…

- Количество итераций каскадной схемы суммирования:

$$k = \log_2 n$$

- Общее количество операций суммирования:

$$\mathrm{K}_{посл} = n/2 + n/4 + \ldots + 1 = n - 1$$

- При параллельном исполнении отдельных итераций каскадной схемы общее количество параллельных операций суммирования является равным:

$$\mathrm{K}_{пар} = \log_2 n$$

Пример: Вычисление частных сумм…

- Показатели ускорения и эффективности каскадной схемы алгоритма суммирования:

$$S_p = T_1 / T_p = (n-1)/\log_2 n,$$
$$E_p = T_1 / pT_p = (n-1)/(p\log_2 n) = (n-1)/((n/2)\log_2 n),$$

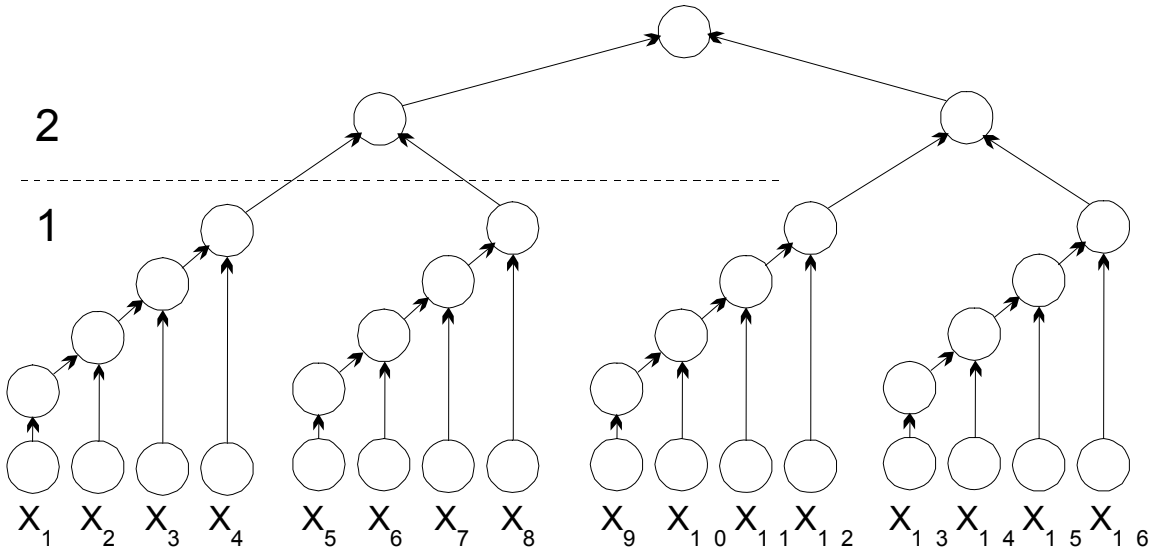
где *p=n/2* есть необходимое для выполнения каскадной схемы количество процессоров.

  - время параллельного выполнения каскадной схемы совпадает с оценкой для паракомпьютера (теорема 2),
  - эффективность использования процессоров уменьшается при увеличении количества суммируемых значений:

$$\lim E_p = 0 \quad при \quad n \to \infty$$

Пример: Вычисление частных сумм…

- **Модифицированная каскадная схема**:
  - Все суммируемые значения подразделяются на *(n/log$_2$n)* групп, в каждой из которых содержится *(log$_2$n)* элементов; для каждой группы вычисляется сумма значений при помощи последовательного алгоритма суммирования;
  - На втором этапе для полученных (n/log$_2$n) сумм отдельных групп применяется обычная каскадная схема:



Пример: Вычисление частных сумм…

- Для выполнения первого этапа требуется $(\log_2 n)$ выполнение параллельных операций при использовании $p = (n/\log_2 n)$ процессоров
- Для выполнения второго этапа необходимо $\log_2(n/\log_2 n) \le \log_2 n$ параллельных операций для $p=(n/\log_2 n)/2$ процессоров
- Время выполнения параллельного алгоритма составляет

$$T_P = 2\log_2 n$$

для $p = (n/\log_2 n)$ процессоров.

Пример: Вычисление частных сумм…

- С учетом полученных оценок показатели ускорения и эффективности модифицированной каскадной схемы определяются соотношениями:

$$S_p = T_1 / T_p = (n-1)/2\log_2 n,$$

$$E_p = T_1 / pT_p = (n-1)/(2(n/\log_2 n)\log_2 n) = (n-1)/2n$$

  - По сравнению с обычной каскадной схемой ускорение уменьшилось в 2 раза,
  - Для эффективности нового метода суммирования можно получить асимптотически ненулевую оценку снизу:

$$E_p = (n-1)/2n \geq 0.25,\ \lim E_p = 0.5 \ при \ n \to \infty.$$

  - Модифицированный каскадный алгоритм является стоимостно-оптимальным (стоимость вычислений пропорциональна времени выполнения последовательного алгоритма):

$$C_p = pT_p = (n/\log_2 n)(2\log_2 n) = 2n$$

Пример: Вычисление частных сумм…

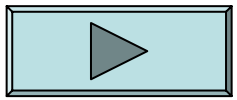
Вычисление всех частных сумм…

- Вычисление всех частных сумм на скалярном компьютере может быть получено при помощи обычного последовательного алгоритма суммирования при том же количестве операций

$$T_1 = n$$

- При параллельном исполнении применение каскадной схемы в явном виде не приводит к желаемым результатам.

Достижение эффективного распараллеливания требует привлечения новых подходов (может быть, даже не имеющих аналогов при последовательном программировании) для разработки новых параллельно-ориентированных алгоритмов решения задач

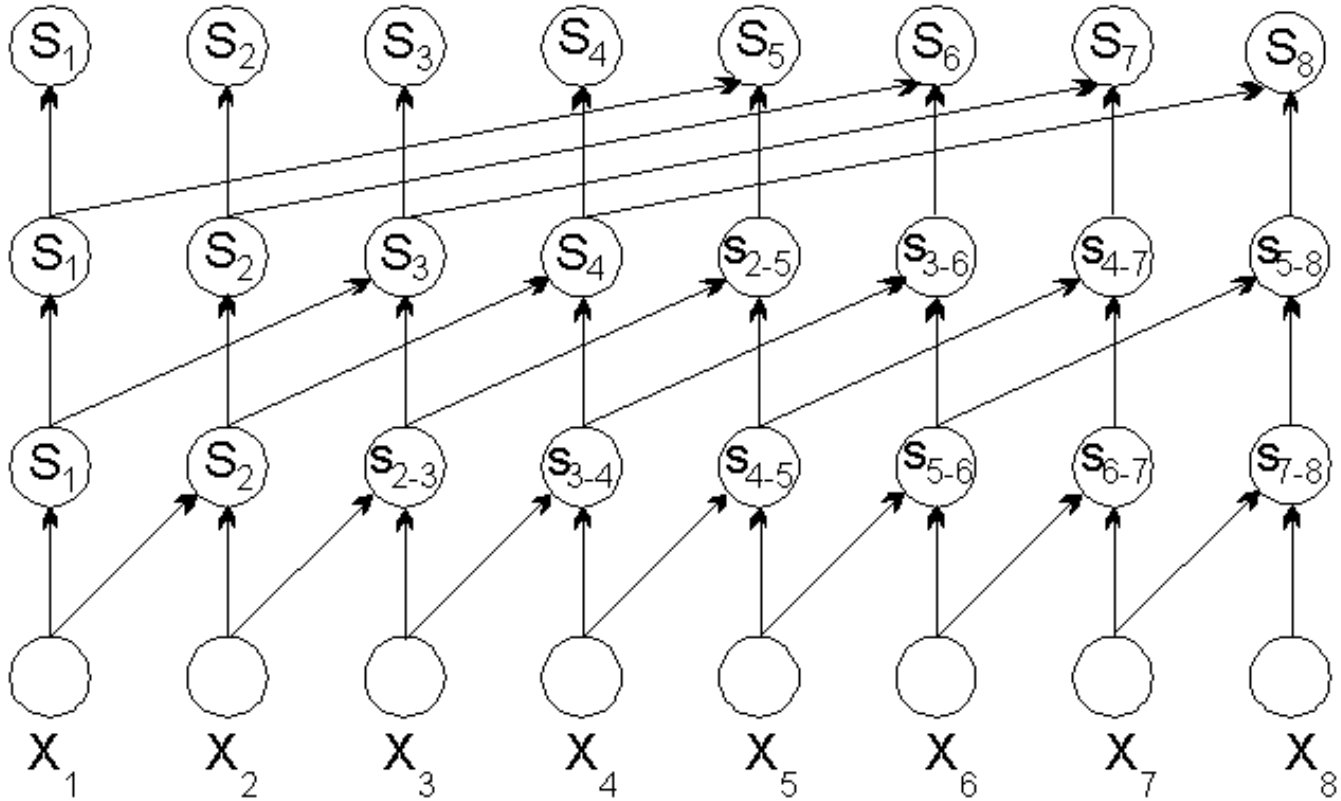


Пример: Вычисление частных сумм…

- **Вычисление всех частных сумм…**
  - Алгоритм, обеспечивающий получение результатов за $log_2 n$ параллельных операций:
    - Перед началом вычислений создается копия S вектора суммируемых значений ($S=x$),
    - Далее на каждой итерации суммирования i, $1 \le i \le log_2 n$, формируется вспомогательный вектор Q путем сдвига вправо вектора S на 2^{i-1} позиций (освобождающиеся при сдвиге позиции слева устанавливаются в нулевые значения); итерация алгоритма завершается параллельной операцией суммирования векторов S и Q.

Пример: Вычисление частных сумм…

- Вычисление всех частных сумм…



Пример: Вычисление частных сумм

- **Вычисление всех частных сумм**
  - Общее количество выполняемых алгоритмом скалярных операций определяется величиной:

$$K_{посл} = n \log_2 n$$

  - Необходимое количество процессоров определяется количеством суммируемых значений:

$$p=n$$

  - Показатели ускорения и эффективности:

$$S_p = T_1 / T_p = n / \log_2 n$$

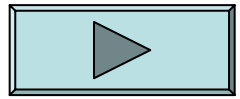
$$E_p = T_1 / pT_p = n / (p \log_2 n) = n / n \log_2 n = 1 / \log_2 n$$

Закон Амдаля. Замечания

- Доля последовательных вычислений может быть существенно снижена при выборе более подходящих для распараллеливания методов,
- Эффект Амдаля

Для большого ряда задач доля последовательных вычислений ***f=f(n)*** является убывающей функцией от ***n***, и в этом случае ускорение для фиксированного числа процессоров может быть увеличено за счет увеличения вычислительной сложности решаемой задачи.

В этом случае, ускорение $S_p = S_p(n)$ является возрастающей функцией от параметра n.



Закон Густавсона – Барсиса…

Оценим максимально достижимое ускорение исходя из имеющейся доли последовательных расчетов в выполняемых параллельных вычислениях:

$$g = \frac{\tau(n)}{\tau(n) + \pi(n)/p}$$

где $\tau(n)$ и $\pi(n)$ есть времена последовательной и параллельной частей выполняемых вычислений соответственно, т.е.

$$T_1 = \tau(n) + \pi(n), \qquad T_p = \tau(n) + \pi(n)/p$$

С учетом введенной величины g можно получить

$$\tau(n) = g \cdot (\tau(n) + \pi(n)/p),\ \pi(n) = (1-g)p \cdot (\tau(n) + \pi(n)/p),$$

что позволяет построить оценку для ускорения

$$S_p = \frac{T_1}{T_p} = \frac{\tau(n) + \pi(n)}{\tau(n) + \pi(n)/p} = \frac{(\tau(n) + \pi(n)/p)(g + (1-g)p)}{\tau(n) + \pi(n)/p}$$

Закон Густавсона - Барсиса

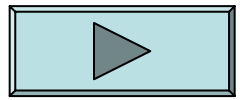
Упрощение последней оценки для ускорения

$$S_p = g + (1-g)p = p + (1-p)g$$

Оценку ускорения, получаемую в соответствии с законом Густавсона-Барсиса, еще называют *ускорением масштабирования* (*scaled speedup*), поскольку данная характеристика может показать, насколько эффективно могут быть организованы параллельные вычисления при увеличении сложности решаемых задач

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Анализ масштабируемости параллельных вычислений...

Параллельный алгоритм называют ***масштабируемым (scalable),*** *если при росте числа процессоров он обеспечивает увеличение ускорения при сохранении постоянного уровня эффективности использования процессоров*



Накладные расходы (*total overhead*) появляются за счет необходимости организации взаимодействия процессоров, выполнения некоторых дополнительных действий, синхронизации параллельных вычислений и т.п.

$$T_0 = pT_p - T_1$$

Новые выражения для времени параллельного решения задачи и получаемого при этом ускорения:

$$T_p = \frac{T_1 + T_0}{p}, \qquad S_p = \frac{T_1}{T_p} = \frac{pT_1}{T_1 + T_0}$$

Тогда эффективность использования процессоров можно выразить как

$$E_p = \frac{S_p}{p} = \frac{T_1}{T_1 + T_0} = \frac{1}{1 + T_0 / T_1}$$

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Анализ масштабируемости параллельных вычислений...

- Если сложность решаемой задачи является фиксированной (T_1=*const*), то при росте числа процессоров эффективность, как правило, будет убывать за счет роста накладных расходов T_0,
- При фиксации числа процессоров эффективность использования процессоров можно улучшить путем повышения сложности решаемой задачи T_1,
- При увеличении числа процессоров в большинстве случаев можно обеспечить определенный уровень эффективности при помощи соответствующего повышения сложности решаемых задач.

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Анализ масштабируемости параллельных вычислений

– Пусть *E=const* есть желаемый уровень эффективности выполняемых вычислений. Из выражения для эффективности можно получить

$$\frac{T_0}{T_1} = \frac{1-E}{E},\ \textit{или}\ T_1 = KT_0,\ K = E/(1-E)$$

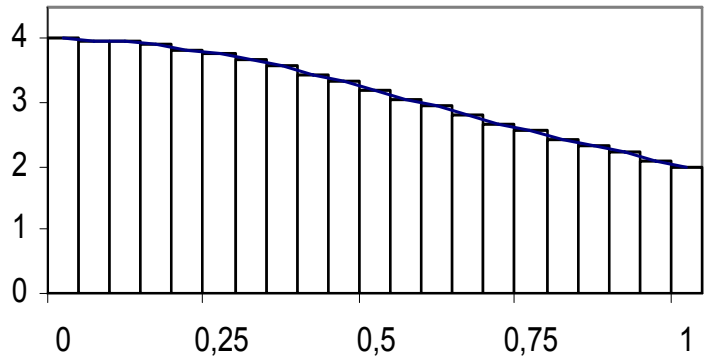
– Порождаемую последним соотношением зависимость *n=F(p)* между сложностью решаемой задачи и числом процессоров обычно называют *функцией изоэффективности* (*isoefficiency function*).

Пример: *Вычисление числа π...*

- Значение числа π может быть получено при помощи интеграла

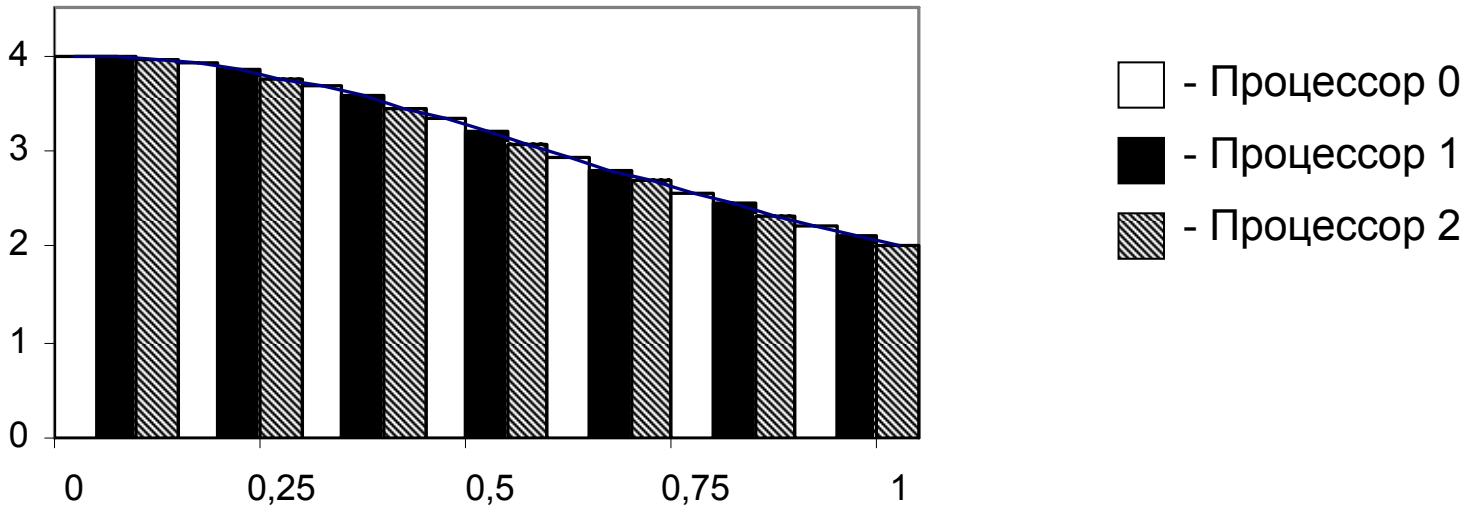
$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

- Для численного интегрирования применим метод прямоугольников



Пример: *Вычисление числа π...*

- Распределим вычисления между ***p*** процессорами (циклическая схема)
- Получаемые на отдельных процессорах частные суммы должны быть просуммированы



Пример: *Вычисление числа π…*

Анализ эффективности…

- n – количество разбиений отрезка [0,1]
- Вычислительная сложность задачи

$$W = T_1 = 6n$$

- Количество узлов сетки на отдельном процессоре

$$m = \lceil n/p \rceil \leq n/p + 1$$

- Объем вычислений на отдельном процессоре

$$W_p = 6m = 6n/p + 6.$$

Пример: *Вычисление числа* π

Анализ эффективности

- Время параллельного решения задачи

$$T_p = 6n/p + 6 + \log_2 p$$

- Ускорение

$$Sp = T_1 / T_p = 6n / (6n/p + 6 + \log_2 p)$$

- Эффективность

$$Ep = 6n / (6n + 6p + p\log_2 p)$$

- Функция изоэффективности

$$W = K(pT_p - W) = K(6p + p\log_2 p)$$

$$\Rightarrow n = [K(6p + p\log_2 p)]/6, \quad (K=E/(1-E))$$

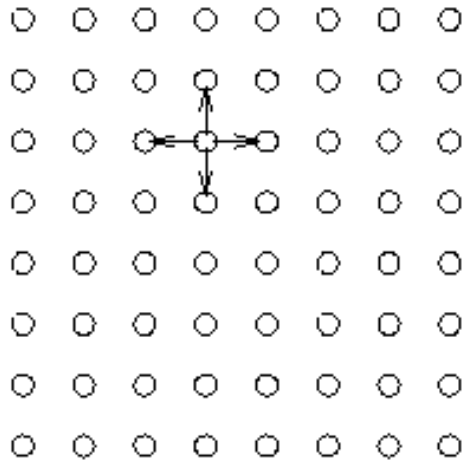
Ex.: E=0.5 при p=8 $\rightarrow n$= 12

при p=64 $\rightarrow n$=128

Пример: *Метод конечных разностей…*

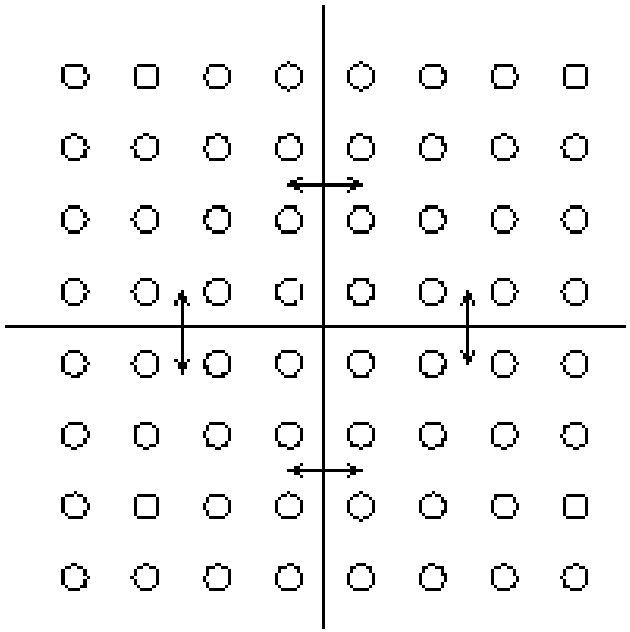
- Метод конечных разностей широко применяется для численного решения уравнений в частных производных (см. раздел 12)
- Рассмотрим схему ($N = 2$)

$$X_{i,j}^{t+1} = w(X_{i,j-1}^{t} + X_{i,j+1}^{t} + X_{i-1,j}^{t} + X_{i+1,j}^{t}) + (1-w) X_{i,j}^{t}$$



Пример: *Метод конечных разностей…*

- Каждый процессор проводит вычисления на прямоугольной подобласти с $(n/\sqrt{p})*(n/\sqrt{p})$ точками
- После выполнения каждой итерации расчета необходима синхронизация расчета



Пример: *Метод конечных разностей*

Анализ эффективности

- $W = T_1 = 6n^2M$ *(M – количество итераций)*
- $T_p = 6n^2M/p + M \log_2 p$
- Ускорение

  $S_p = T_1 / T_p = 6n^2 / (6n^2/p + \log_2 p)$
- Функция изоэффективности

  $W = K(pT_p - W) = K(p\log_2 p)$

  $\Rightarrow n^2 = [K(p\log_2 p)]/6, \quad (K=E/(1-E))$

Метод конечных разностей является более масштабируемым, чем метод прямоугольников

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Выводы

1. При разработке параллельных алгоритмов принципиальное значение имеет получение оценок:
   - Максимально-возможного ускорения решения задачи,
   - Ускорения, которое может быть получено для отдельных алгоритмов
2. Данные оценки могут быть получены на этапе исследования (разработки) алгоритмов
3. Основа для получения таких оценок – анализ графа информационных зависимостей задачи
4. Разработка параллельных алгоритмов может потребовать создания принципиально новых методов решения задач

МОДЕЛИ ВЫЧИСЛЕНИЙ.
Рекомендуемая литература

В.В.Воеводин, Вл.В.Воеводин. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002.- 608 с.

Заключение

1. При разработке параллельных алгоритмов принципиальное значение имеет получение оценок:
   - Максимально-возможного ускорения решения задачи,
   - Ускорения, которое может быть получено для отдельных алгоритмов
2. Данные оценки могут быть получены на этапе исследования (разработки) алгоритмов
3. Основа для получения таких оценок – анализ графа информационных зависимостей задачи
4. Разработка параллельных алгоритмов может потребовать создания принципиально новых методов решения задач

Контакты:

Нижегородский университет

Факультет вычислительной математики и кибернетики

Гергель Виктор Павлович

gergel@unn.ru

СПАСИБО ЗА ВНИМАНИЕ

Вопросы ?