Введение в параллельное программирование в системах с общей памятью

Материал для дополнительных занятий
по дисциплине «Программирование»,
изучаемой студентами 1-ого курса механико-математического факультета НГУ

Занятие №1 (вводное)

I. Введение. Стандартный путь начинающего многопоточного программиста. Пример программы

(семинар, 2 часа)

Заголовки слайдов (1)

1. «Обычная история» начинающего многопоточного программиста
2. Условимся об обозначениях...
3. Вы наметили план действий...
4. Вы создали тест - последовательный код умножения матриц...
5. Определили независимые вычисления в умножении матриц...
6. Программист N** сказал, что это... простейший способ параллельного вычисления строк новой матрицы...
7. И у Вас возникло много вопросов...
8. Что означает

```
#pragma omp parallel for private(j, k)...
```

Заголовки слайдов (2)

9. Но Вы все-таки пытаетесь уточнить: а что же делает каждый поток?
10. Вы добиваетесь ясности – с какими переменными каждый поток выполняет три «кратные» цикла
11. А у Вас возник вопрос – как потоки делят итерации цикла...
12. Параллельная схема №1(для двух потоков)
13. Параллельная схема №2 (для двух потоков)
14. Способ «дележа итераций» неизвестен...
15. Ваш следующий вопрос –какие переменные «общие» ?
16. Ваша реакция на ЧАСТНЫЕ и ОБЩИЕ – «понятно, но смутно...»
17. А как программу «запускать» и будет ли программа работать... быстрее?

Заголовки слайдов (3)

18. Кроме того, программист N** добавил...
19. Ваша реакция...
20. Реакция программиста N** на Вас...
21. Вы решили начать с установок проекта...
22. Установка поддержки директив OpenMP (слайд 1)
23. Установка поддержки директив OpenMP (слайд 2)
24. Настройка на многопоточно-безопасные библиотеки
25. Какие же «параллельные ошибки» могут создать не те библиотеки ?
26. Конфликты памяти: «своя память» и «общая»
27. Конфликты памяти или гонки данных
28. А почему все же «дорогое удовольствие» - создать поток, зачем БОЛЬШАЯ матрица...

Заголовки слайдов (4)

29. Вы перешли к следующему пункту – как «ужиться» с оптимизацией и что это такое...
30. И программист N** привел два примера крупноблочного параллелизма...
31. Вначале, сказал программист N** , нужно модернизировать последовательный код...
32. Затем –изменить функцию так, чтобы она вычисляла лишь группу строк... (схема №1)
33. Заключительный этап – вставить «прагму»... (схема №1)
34. Но, сказал программист N** , это не лучший вариант крупноблочного распараллеливания... (схема №1)
35. Для параллельной схемы №2 потребуются более существенные изменения...

Заголовки слайдов (6)

36. И тут возник вопрос – что важнее – параллелизм или оптимизация…
37. У Вас – «переполнение стека» - близок ли конец этим смежным вопросам…
38. Ответ программиста N** : вообще-то да… забыл, но.. не берись за все сразу
39. Советы программиста N** о последовательности обучения параллельному программированию…
40. Выводы

Цель занятия

- На примере «жизненной истории»
  - Выполнить обзор факторов, существенных для правильной и быстрой работы многопоточной программы
    - Не так уж часто бывает, что у новичка есть «старший товарищ», которому можно задавать «дурацкие вопросы»
  - Показать последовательность изучения материала

«Жизненная история»

- На новом месте работы Вам предложили
  - самостоятельно научиться программировать в системах с общей памятью
  - Вам в помощь предложили имеющего опыт программиста N**, который
    - Может ответить на Ваши вопросы
- Рабочее место программиста N** рядом с Вашим. Он что-то программирует у себя на компьютере, а на Ваши вопросы отвечает короткими репликами, не всегда понятными
  - Вам часто приходиться уточнять, а что имел ввиду программист N**

1. «Обычная история» начинающего многопоточного программиста

- Вы решили изучить многопоточное программирование в *системах с общей памятью* ... *максимально быстро!*
- Большинство программистов лучше всего *«понимают»* ... *на практике* и Вы не исключение!
- Вы выбрали
  - простую задач – умножение матриц
  - параллельную технологию с самым лаконичным синтаксисом – *стандарт OpenMP*
- Вы решили
  - сделать параллельную реализацию задачи
  - заодно на практике *«наступить на все грабли»!*

2. Условимся об обозначениях…

- Предположим, что в Вашем распоряжении имеется
  - либо многопроцессорная машина с общей памятью
  - либо многоядерная машина
- но *Вы незнакомы*
  - с архитектурой
  - с операционной системой
- *Знаете лишь* то, что она… *может считать параллельно*
- Параллельных «исполнителей» (для нас – пока что неизвестные внутренние объекты системы) назовем
  - поток 1
  - поток 2
  - поток …

3. Вы наметили план действий…

- Вы знаете язык программирования С
- Вы умеете работать в среде Visual Studio.Net
- Вы умеете создавать проект «консольное приложение C++»

- Сначала Вы решили
  - *написать тест* - обычную последовательную программу умножения матриц

- Затем Вы решили
  - *выделить* в задаче те *вычисления,* которые выполняются *независимо – значит, можно параллельно!*

- После этого Вы решили
  - Определить более конкретно те задания, которые «параллельные исполители» – потоки – будут выполнять параллельно…

4. Вы создали тест - последовательный код умножения матриц…

```
int main()
{
   …
 for ( i = 0; i < N; i++)
  for ( j = 0; j < N; j++)
   for ( k = 0; k < N; k++)
     c[i][j] = c[i][j] + a[i][k] * b[k][j];
 …
}
// итерация внешнего цикла – одна строка новой
// матрицы
```

5. Определили независимые вычисления в умножении матриц...

- C = A * B
- C[i,j] = $\sum$ A[i,k] * B[k,j]
  - Элемент матрицы произведения равен скалярному произведению вектора - строки первой матрицы на вектор – столбец второй
- Вычисления каждого элемента матрицы произведения независимы и могут выполняться параллельно
  - Вычисления группы строк матрицы произведения независимы
    - итерации внешнего цикла можно выполнять параллельно

6. Программист N** сказал, что это… простейший способ параллельного вычисления строк новой матрицы…

```
int main()
{
  …
#pragma omp parallel for private(j, k)
 for ( i = 0; i < N; i++)
  for ( j = 0; j < N; j++)
   for ( k = 0; k < N; k++)
     c[i][j] = c[i][j] + a[i][k] * b[k][j];
 …
}
// итерация внешнего цикла – одна строка новой
// матрицы
```

7. И у Вас возникло много вопросов…

- А что вообще означает строчка
  - `#pragma omp parallel for private(j, k)` **???**
- А будет ли программа *работать быстрее*, если я просто вставлю эту *«магическую»* строчку, а потом отправлю на компиляцию и выполнение?

8. Что означает `#pragma omp parallel for private(j, k)…`

- **`#pragma omp parallel for`** – это директива для *параллельного выполнения итераций* цикла for количеством N
  - *Количество «параллельных исполнителей» - потоков* устанавливается равным **`n_thread`** - *числу процессоров или ядер* (если кроме этой директивы нет больше ничего)
  - Каждый поток выполняет **`N/ n_thread`** итераций

- **`private(j, k)`** – это *дополнение* к директиве, так называемая *«клауза» (**`clause`**)*
  - **`private(j, k)`** – означает, что переменные **`j, k`** являются ЧАСТНЫМИ переменными для каждого потока
    - **`j, k`** у различных «параллельных исполнителей» - потоков могут принимать разные значения; ни один поток не знает, чему равны значения этих переменных у соседа

9. Но Вы все-таки пытаетесь уточнить: а что же делает каждый поток?

- А программист N** отвечает, что OpenMP реализует так называемый *«параллелизм по данным»*
  - Каждый поток выполняет одинаковый код
    - В данном случае – три «кратных» цикла
- А вот с переменными в этом участке дело обстоит по-разному:
  - Часть переменных поток считает своей *«личной кухней»*
    - и делает с ними что хочет, не оповещая соседей
      - это ЧАСТНЫЕ переменные
  - А вот с другими переменными дело обстоит иначе:
    - *если один* поток *изменил* значение переменной – *все* остальные потоки тоже *«принимают для себя»* это значение
      - это ОБЩИЕ переменные

10. Вы добиваетесь ясности – с какими переменными каждый поток выполняет три «кратные» цикла

- Программист N** отвечает, что:
  - Для 0-ого потока k, j изменяются от 0 до N-1
  - Для 1-ого потока k, j изменяются от 0 до N-1
  - …
  - Для n_thread-1 потока k, j изменяются от 0 до N-1
    - Например, k в 0-ом потоке может быть 1, а в 1-ом 2 в одно и то же время, аналогично j – они ЧАСТНЫЕ
    - k в 0-ом потоке может быть 1, а в 1-ом тоже 1

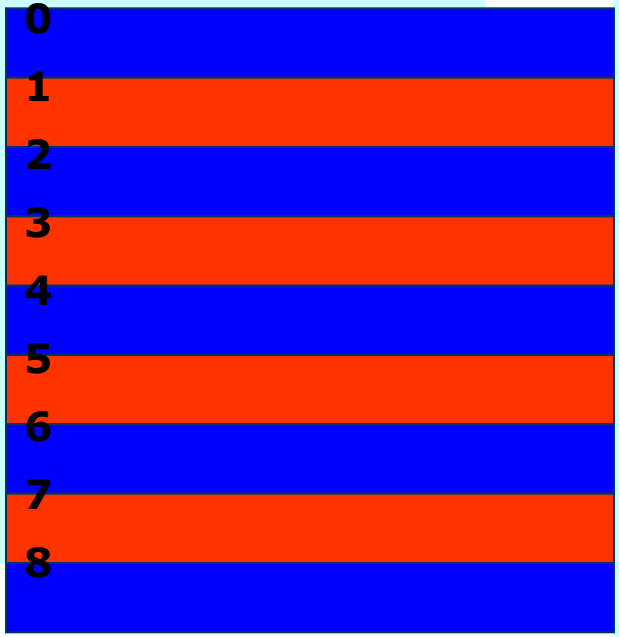
- А вот диапазон значений переменной внешнего цикла i делится на части между различными потоками
  - Значение i во всех потоках разные- i тоже ЧАСТНАЯ
    - Если потоков два, то в одном все значения i могут быть четные, а в другом – все нечетные

11. А у Вас возник вопрос – как потоки делят итерации цикла…

- Возможно множество вариантов, например
  - Параллельная схема №1. Поток с номером ID последовательно выполняет итерации
    - 1-ая итерация i = ID (ID = 0,1,2..n_thread-1)
    - 2-ая итерация i = ID + n_thread
    - 3-я итерация i = ID + 2 * n_thread … и. т. д.
  - Параллельная схема №2. Поток с номером ID выполняет непрерывный диапазон значений i от i1 до i2
    - от i1 = ID * (N / n_thread)
    - до i2 = (ID +1)* (N / n_thread) -1
      - i2 = n_thread (если ID = n_thread - 1)
        - N – размер матрицы

12. Параллельная схема №1 (для двух потоков)

- Поток с номером ID = 0 (**синий цвет**) последовательно выполняет итерации
  - 1-ая итерация i = 0
  - 2-ая итерация i = 2
  - 3-я итерация i = 4
    - и т. д.
- Поток с номером ID = 1 (**красный цвет**) последовательно выполняет итерации
  - 1-ая итерация i = 1
  - 2-ая итерация i = 3
  - 3-я итерация i = 5
    - и т. д.

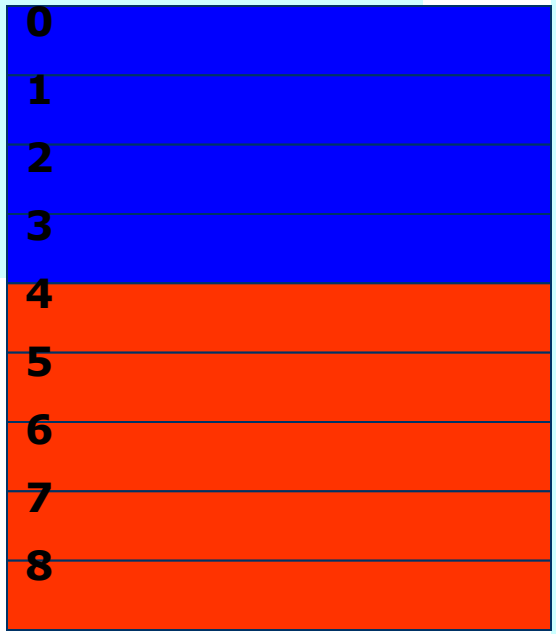


13. Параллельная схема №2 (для двух потоков)

- Параллельно выполняются вычисления группы строк матрицы (номер группы равен номеру потока ID) с индексами в диапазоне
  - от i1 = ID * (N / n_thread)
  - до i2 = (ID +1)* (N / n_thread) -1
  - до i2 = n_thread (ID=n_thread-1)
    - N – размер матрицы

< Группы строк одного цвета вычисляет один поток с номером ID

<Поток №0 – **синие** строки
<Поток №1 – **красные** строки



14. Способ «дележа итераций» неизвестен…

- Но, к сожалению программист N** сказал, что
  - Неизвестно, по какой из двух параллельных схем (по №1 или по №2) происходит раздел итераций цикла for между потоками
  - Выбранная компилятором схема зависит от внутренних настроек OpenMP
    - Возможна параллельная схема №1
    - Возможна параллельная схема №2
- Конфликт OpenMP с оптимизацией в любом случае роли не сыграет, так как на каждой итерации внешнего цикла сравнительно много вычислений

15. Ваш следующий вопрос –какие переменные «общие» ?

- Все переменные, не указанные в **`private()`**, исключая переменную цикла (она «частная» согласно директиве), являются ОБЩИМИ или РАЗДЕЛЯЕМЫМИ (SHARED)
  - Если один поток изменяет значение разделяемой переменной, то этим измененным значением пользуется любой поток, которому она понадобилась
- Матрица **`A`** является ОБЩЕЙ для всех потоков, а все индексы – **`i,j,k`** – ЧАСТНЫЕ переменные каждого потока, для каждого потока – своя часть матрицы
  - Есть возможность пронаблюдать параллельную ошибку – уберите из директивы «хвост» **`private(j, k)`**
    - Ошибка относится к типу «конфликт памяти» - поместили переменную не в «частную память» каждого потока, а в «общую» для всех…

16. Ваша реакция на ЧАСТНЫЕ и ОБЩИЕ – «понятно, но смутно...»

- Программист N Вам отвечает, что
  - ясность, увы, появляется только на практике
- А Вы, в свою очередь, спрашиваете...
  - Как же эту параллельную программу отправить на выполнение?
  - Вставил «прагму», откомпилировал и выполнять?
  - И ускорится во столько раз, сколько в системе процессоров/ядер ?

17. А как программу «запускать» и будет ли программа работать… быстрее?

- А программист N ответил, что…
  - Может быть все, что угодно:
    - Замедлится
    - Ускорится примерно в n_thread раз
    - Ускорится в меньшее количество раз
    - Будет работать правильно
      - или неправильно
  - Многое зависит от установленных вами свойств проекта
  - А что-то может зависеть от версии компилятора
  - Но при малом размере матрицы работать быстрее точно не будет…

18. Кроме того, программист N добавил…

- Скорее всего, ваша ПРОГРАММА ЗАМЕДЛИТСЯ
- Но, если Вы...
  - Правильно выберете установки проекта в Visual Studio.Net
  - Установите достаточно большой размер матрицы
- ТОГДА, скорее всего УСКОРИТСЯ...
- Но если бы на одной итерации цикла было меньше вычислений, ЛУЧШЕ эту программу переписать ПО-ДРУГОМУ…

19. Ваша реакция…

- Зачем дополнительные установки в проекте – ведь, по общему виду строчка с **`#pragma`** … – всего лишь директива компилятора?
- Почему же программа может работать неправильно?
- Как переписать по-другому, чтобы не «убить» оптимизацию?
- И почему матрицы должны быть очень большими?
- Что ВООБЩЕ за наворотЫ – ведь ты, программист N**, говорил «ВСЕ ПРОСТО!!!»

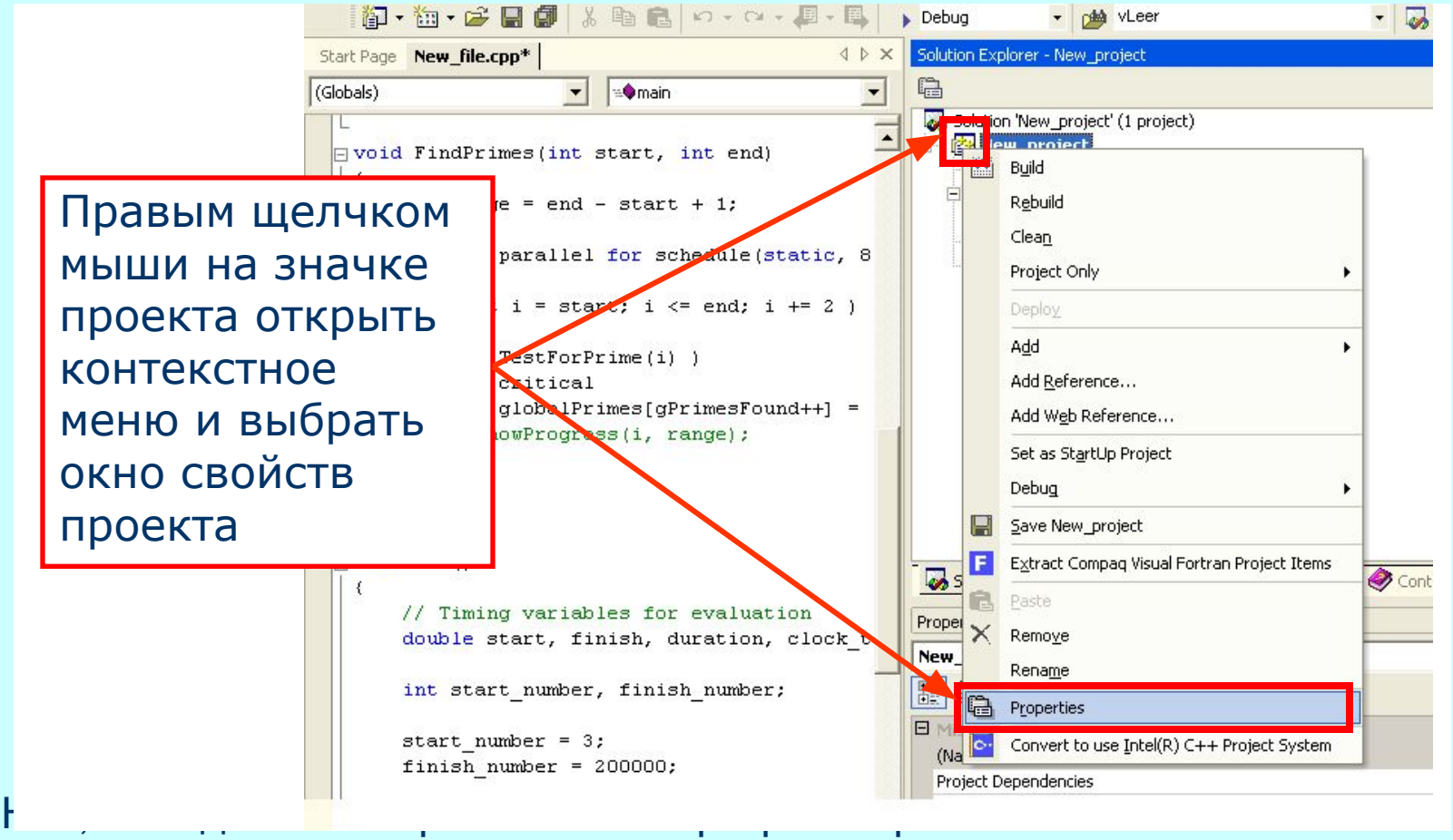
20. Реакция программиста N** на Вас…

- Ну, конечно просто, надо только ПРИВЫКНУТЬ…
- Стандарт OpenMP является ДОПОЛНИТЕЛЬНЫМ классом прагм для компилятора C++, поддержку этого класса прагм нужно устанавливать ДОПОЛНИТЕЛЬНО
- Нужно подключить динамические библиотеки, которые поддерживают одновременную работу нескольких потоков и не плодят «параллельных ошибок»…
- Зачем большая матрица – так ведь вызвать на работу параллельного исполнителя – *дорогое «по времени и пространству» удовольствие*, вызывать нужно только «на большую работу»…
- Ну а насчет конфликта с оптимизацией – долго объяснять, в этом случае это не играет роли, хотя при «легковесных итерациях» лучше не рисковать… и предполагать по-другому…

21. Вы решили начать с установок проекта…

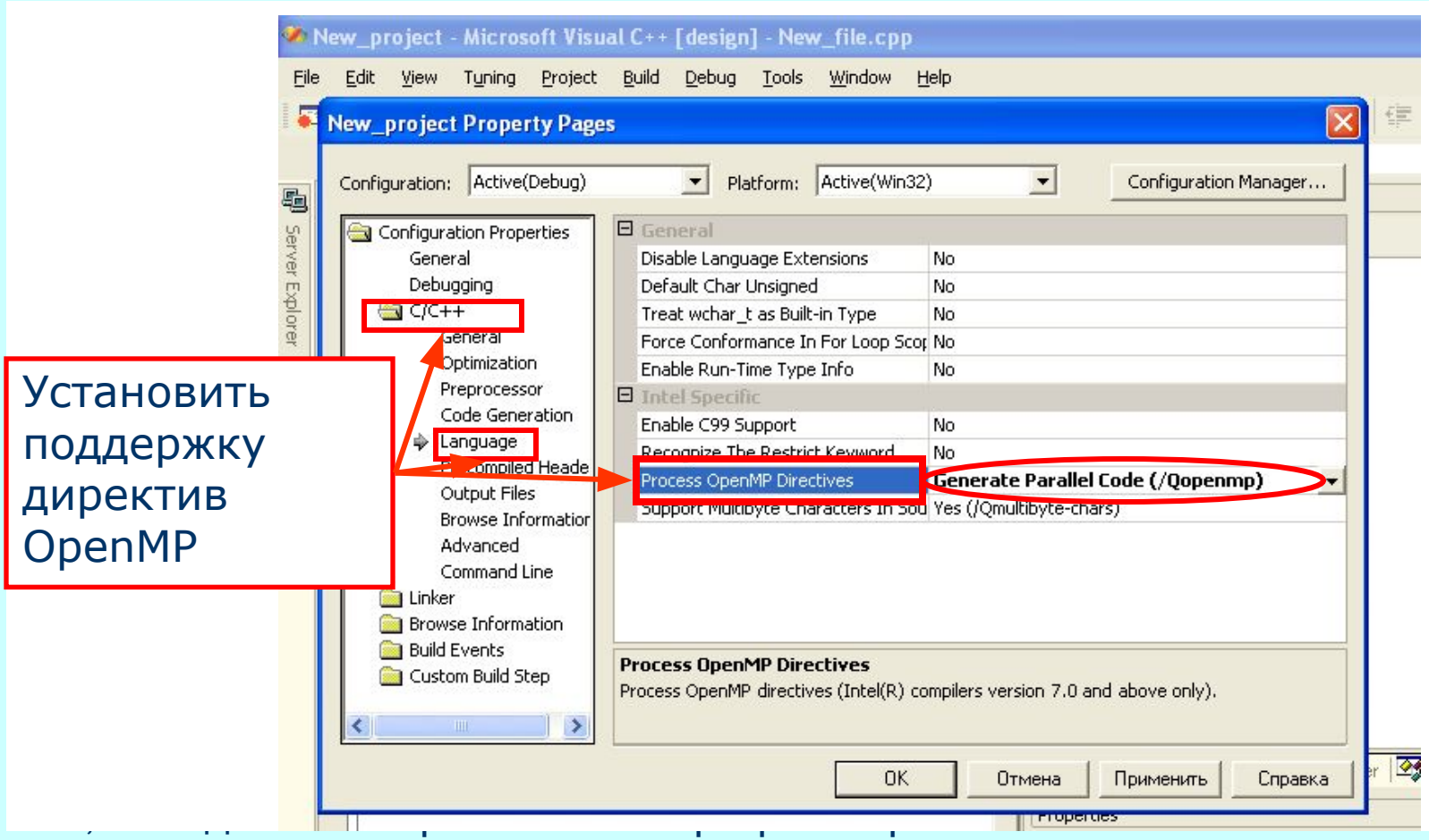
- Установить в проекте поддержку дополнительного класса прагм – стандарта OpenMP:
  - **Properties>> C/C++>> Language>> Process OpenMP Directives>> Generate Parallel Code (/Qopenmp)**
- Подключить «параллельных исполнителей» - многопоточно – безопасные динамические библиотеки…
  - **Properties>> C/C++>> Code Generation>> Runtime Library>> Multi-threaded Debug DLL(/MDd) (или для Release: Multi-threaded DLL(/MDd))**
  - Это оказалось не так уж трудно…

22. Установка поддержки директив OpenMP (слайд 1)

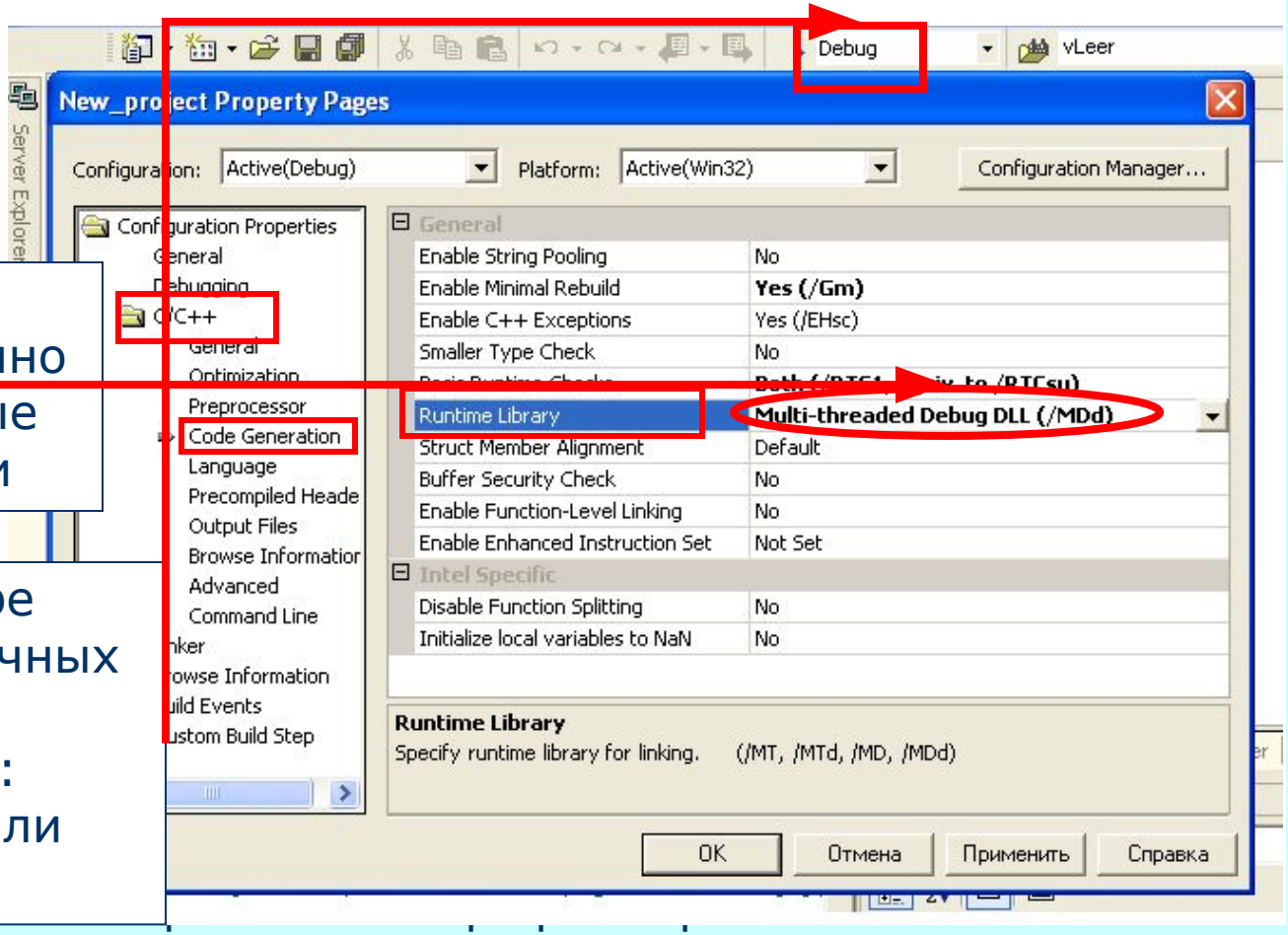


системах с общей памятью»
(ММФ, 1 курс, «Программирование»)

23. Установка поддержки директив OpenMP (слайд 2)



24. Настройка на многопоточно-безопасные библиотеки



Выбрать многопоточно-безопасные библиотеки

При выборе многопоточных библиотек учитывать: «Debug» или «Release»

25. Какие же «параллельные ошибки» могут создать не те библиотеки ?

- На что программист N** ответил, что, существует два больших класса ошибок:
  - Конфликты памяти
  - Различные зависания потоков на конечное и бесконечное время
- Но Вам, как занимающемуся только таким параллелизмом, где распределение заданий для потоков заранее известно (ЯВНЫЙ ПАРАЛЛЕЛИЗМ), актуальны
  - Конфликты памяти или гонки данных
    - Кроме того, один поток направляется на один процессор/ядро – не будет ускорения

26. Конфликты памяти: «своя память» и «общая»

- В многопоточно-безопасных библиотеках потоки хорошо различают «свою память» (частную, private) и «общую память» (разделяемую, shared)
- В частной памяти потоки работают независимо друг от друга с частными переменными
  - У разных потоков частная переменная может иметь различные значения
- В общей памяти потоки по очереди изменяют значения общих переменных
  - Задание и чтение значений общих переменных для всех потоков

27. Конфликты памяти или гонки данных

- А в библиотеках, предназначенных для одного потока, все переменные общие (разделяемые) и вся память общая (разделяемая)
  - Результат любой переменной один для всех потоков
  - Например, первый поток выполняет: a = 2;
  - А второй поток выполняет: a = 3;
    - Результат зависит от того, какой поток успеет вперед – возникла «гонка данных»
- Программист N** добавил, что при использовании OpenMP эта ошибка возникает нередко

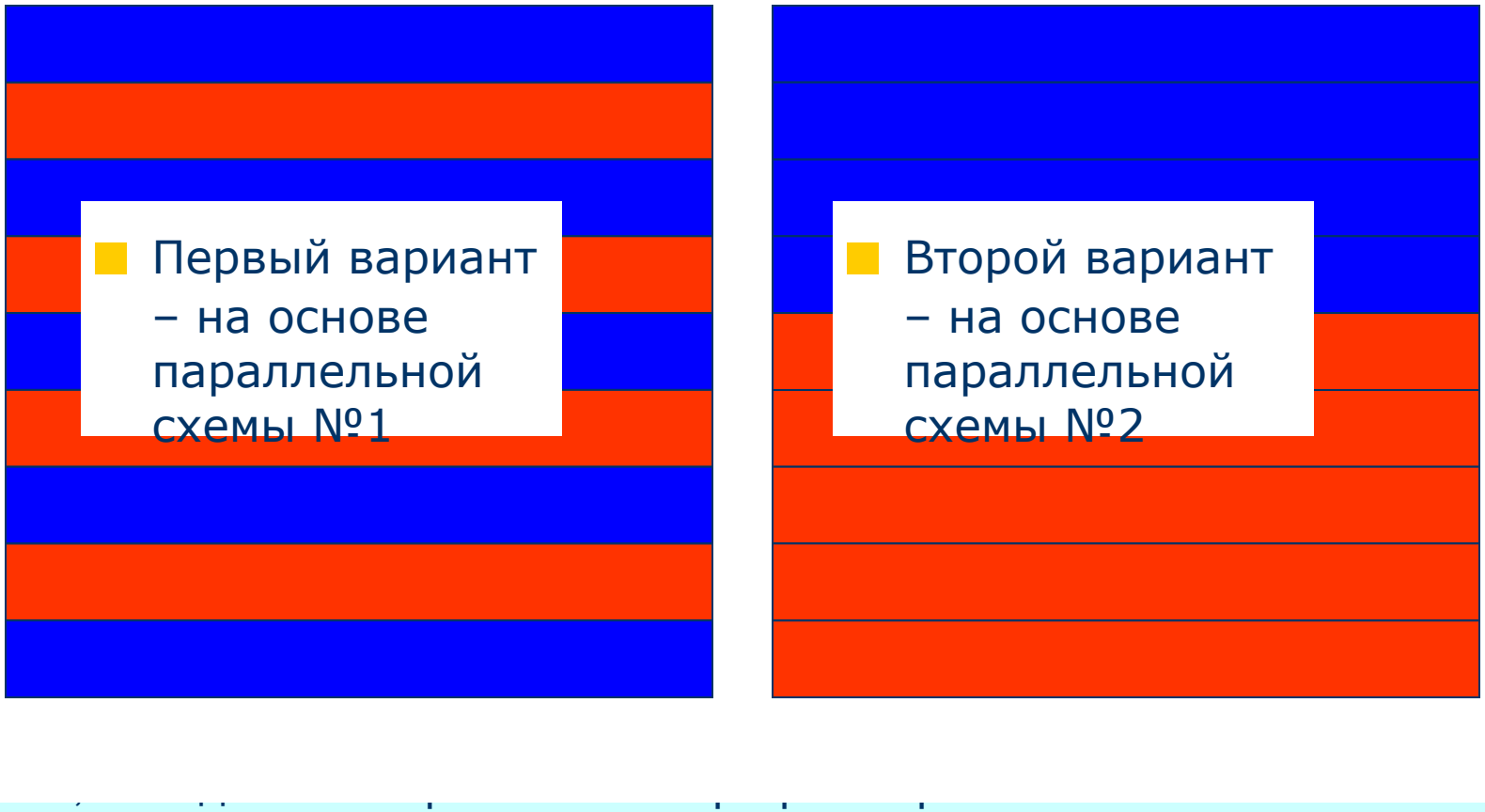
28. А почему все же «дорогое удовольствие» - создать поток, зачем БОЛЬШАЯ матрица…

- Программист N** ответил Вам, что в операционной системе Windows
  - Поток – это объект ядра операционной системы,
    - Чтобы создать поток, требуется «создать объект ядра»
    - На создание чего-то в ядре требуется время
    - А еще потоку требуется память
  - Все процессы (каждый процесс имеет свои потоки, Ваш процесс – это умножение матриц) где-то через 25 миллисекунд уступают «место на процессоре» соседу,
    - так что ПОЛНОЕ ВРЕМЯ параллельных вычислений должно быть больше этого кванта в 25 милисекунд

29. Вы перешли к следующему пункту – как «ужиться» с оптимизацией и что это такое…

- Программист N** сказал, что оптимизация программы выполняется на уровне компилятора
  - По различным параметрам, например:
    - По времени выполнения
    - По объему занимаемой памяти
  - Как правило, оптимизируется достаточно простой код,
    - В котором компилятор не видит «зависимости по данным» (для цикла – вычисления на разных итерациях взаимозависимы)
  - «прагма» может создать для компилятора видимость «зависимости по данным»
    - Код станет «непригодным» для оптимизации
- Беспроигрышный вариант для ЯВНОГО ПАРАЛЛЕЛИЗМА – крупноблочное распараллеливание

30. И программист N** привел два примера крупноблочного параллелизма…



системах с общей памятью»
(ММФ, 1 курс, «Программирование»)

31. Вначале, сказал программист N**, нужно модернизировать последовательный код…

```
int ThreadFunc() //1-ый этап
{int i,j, k;
for ( i = 0; i < N; i++)
 for ( j = 0; j < N; j++)
  for ( k = 0; k < N; k++)
    c[i][j] = c[i][j] + a[i][k] * b[k][j]
return 0;
}

int main()
{
 …
 ThreadFunc();
…
}
```

32. Затем –изменить функцию так, чтобы она вычисляла лишь группу строк… (схема №1)

```
int ThreadFunc1(int ID, int n_thread) // 2-ой этап
{int i, j, k;
for ( i = ID; i < N; i+ = n_thread)
 for ( j = 0; j < N; j++)
  for ( k = 0; k < N; k++)
    c[i][j] = c[i][j] + a[i][k] * b[k][j]
return 0;
}

int main()
{
 …
 for ( i = 0; i < n_thread; i++) ThreadFunc1(
   i,n_thread );
}
```

33. Заключительный этап – вставить «прагму»… (схема №1)

```
int ThreadFunc1(int ID, int n_thread) // 3-ий этап
{int i, j, k;
for ( i = ID; i < N; i+ = n_thread)
 for ( j = 0; j < N; j++)
  for ( k = 0; k < N; k++)
    c[i][j] = c[i][j] + a[i][k] * b[k][j]
return 0;
}
#pragma omp parallel for
int main()
{
 for ( i = 0; i < n_thread; i++) ThreadFunc1( i,
   n_thread );
}
```

34. Но, сказал программист N** , это не лучший вариант крупноблочного распараллеливания… (схема №1)

- В цикле с шагом, равным переменной **n_thread**, компилятор может «заподозрить» зависимость по данным

```
for ( i = ID; i < N; i+ = n_thread)
 for ( j = 0; j < N; j++)
  for ( k = 0; k < N; k++)
    c[i][j] = c[i][j] + a[i][k] * b[i][j]
```

- «Подозрения» компилятора беспочвенны, но
  - Компилятор может отказаться оптимизировать цикл
    - Проблему может и можно решить с помощью еще одной компиляторной директивы, но, повторяю, здесь оптимизация внешнего цикла не актуальна

35. Для параллельной схемы №2 потребуются более существенные изменения…

```
int ThreadFunc2(int ID, int n_thread)
{int i, j, k;
int Initial=ID*N/n_thread, Final=(ID+1)*N/n_thread;
if(ID==n_thread-1){Final=N};
for ( i = Initial; i < Final; i++)
 for ( j = 0; j < N; j++)
  for ( k = 0; k < N; k++)
    c[i][j] = c[i][j] + a[i][k] * b[i][j]
return 0;
}
int main()
{
#pragma omp parallel for
 for ( i = 0; i < n_thread; i++) ThreadFunc2( i,
   n_thread );
}
```

36. И тут возник вопрос – что важнее – параллелизм или оптимизация…

- Программист N** заметил, что вообще-то бы хорошо все использовать, а конкретно для оптимизации умножения матриц компилятором Intel C++ на двухъядерной машине коэффициенты ускорения примерно следующие:
  - В 2 раза – за счет параллелизма
  - В 2 раза – за счет оптимизации работы с памятью методом модернизации кода (для больших матриц)
  - В 20 раз – выбор правильных ключей оптимизации
    - Ну в 80 раз ускорить можно тупо, а при желании и до 100 догнать

- Ну это для умножения матриц, а если, например, на каждой итерации цикла по условному оператору, параллелизм сравним с оптимизацией

37. У Вас – «переполнение стека» - близок ли конец этим смежным вопросам…

- И все ли важное для ускорения программы, ты, программист N**, перечислил:
  - Затраты времени и памяти на создание потоков
  - Конфликт OpenMP и оптимизации
  - Настройки проекта:
    - Подключение OpenMP
    - Многопоточно-безопасные библиотеки
  - Параллельные ошибки…
- Или что-то еще забыл?

38. Ответ программиста N** : вообще-то да… забыл, но.. не берись за все сразу

- Программист N сказал, что есть правда, проблема, когда один поток все сделал и спит, а другой вкалывает, но
  - В перемножении матриц это несущественно
  - Проблема называется дисбалансом потоков
- А вообще, я тебе просто дал обзор всего, что необходимо, чтобы ускорить программу, но
  - Учить все сразу бесполезно
  - Каждая проблема – большая область для специалиста
- Есть обучение методом «положительного подкрепления»: учишь что-то одно, а на остальное – нуль внимания

39. Советы программиста N** о последовательности обучения параллельному программированию…

- Отключи-ка ты для начала всю оптимизацию и добейся хотя бы того, чтобы не было «параллельных ошибок»
  - На скорость внимания вообще не обращай
  - Потом постарайся выбрать параллельную схему так, чтобы при отсутствии оптимизации она ускорялась
    - Оцени алгоритмически допустимое ускорение
      - Выбери распределение заданий «параллельным исполнителям» «поровну»
      - Определи экспериментально параметры задачи, когда выгода от параллелизма выше затрат на его создание
- Ну а потом подключи оптимизацию и корректируй параллельную схему «методом тыка» «на скорость»
- Когда надоест вслепую запускать и наберешь эмпирический опыт, поневоле тебе захочется почитать что-нибудь про архитектуру и операционную систему…
  - Тогда и поймешь, когда почувствуешь, что НАДО…

40. Выводы

- Показан пример кода простой параллельной программы на примере умножения матриц
- Показаны факторы, существенные для правильной и быстрой работы программы
- Предложена последовательность обучения параллельному программированию в системах с общей памятью …
  - Программист N** : «Главное – привыкнуть к параллелизму…»