МНОГОПОТОЧНОСТЬ И ПАРАЛЛЕЛИЗМ В C++

Курсы Intel Delta-3

Алексей Куканов, Intel Corporation

Нижний Новгород, 2015

Часть 3. Понемногу о разном

Настройка Intel® Threading Building Blocks

- Загрузите пакет с www.threadingbuildingblocks.org
  - Например, tbb43_20150209oss_win.zip
- Распакуйте содержимое в директорию
  - Например, в C:\Intel\TBB4.3
- Настройте окружение и/или IDE

- В командной строке для Visual C++:
  - `C:\Intel\TBB4.3\tbbvars.bat` *`ia32 vc12`*
  - Для получения списка платформ и версий VS: `C:\Intel\TBB4.3\tbbvars.bat`
- Для сборки приложения:
  - `cl /EHsc` *`<other options>`*

Настройка проекта в Visual Studio

- Откройте свойства проекта
- → C/C++ → Additional Include Directories
  - Добавьте C:\Intel\TBB4.3\include
- → Linker → Additional Library Directories
  - Добавьте C:\Intel\TBB4.3\lib*ia32**vc12*
- → Debugging → Environment
  - Добавьте PATH=%PATH%;C:\Intel\TBB4.3\bin*ia32**vc12*

Настройка для MinGW

- Загрузите пакет с исходными файлами
  - Например, tbb43_20150209oss_src.zip
- Распакуйте содержимое в директорию
  - Например, в C:\Intel\TBB4.3
- В командной строке для MinGW:
  - `cd C:\Intel\TBB4.3`
  - `gmake compiler=gcc`
  - `gmake –C src compiler=gcc tbbvars`
  - `build\windows_ia32_gcc_mingw4.8_release\tbbvars.bat`
- Для сборки приложения:
  - `g++ <other options> -ltbb`

Обзор функционала Intel® TBB

Параллельные алгоритмы

Высокоуровневый, эффективный, масштабируемый и компонуемый подход к параллельному программированию

Потокобезопасные контейнеры

Одновременный доступ, масштабируемость, совместимость с STL (в пределах возможного)

Граф вычислений (Flow Graph)

Набор классов для построения графа вычислений с зависимостями по данным или по управлению

Данные, локальные для потока

Без ограничений на число переменных, с возможностью обхода

Планировщик работы

«Продвинутый» механизм исполнения задач для эффективной реализации параллельных алгоритмов

Примитивы синхронизации

Атомарные переменные, мьютексы с разными свойствами

Прочее

Потоки, таймер, исключения

Управление памятью

Менеджер памяти и C++ аллокаторы с разными свойствами

Будущее не за горами

- Тема *concurrency* и параллелизма активно исследуется в комитете по стандарту C++
- Выпущены / в разработке технические спецификации:
  - TS for C++ Extensions for Parallelism: параллельные STL алгоритмы
  - TS for C++ Extensions for Concurrency: развитие std::future
  - TS for C++ Extensions for Transactional Memory
- Прочие предложения к стандартизации
  - Language Extensions for Vector loop level parallelism
  - Task Region (шаблон fork-join)
  - Concurrent containers
  - Многие другие
- Следующая версия стандарта (C++17) наверняка будет включать многое из вышеуказанного

ЗАКЛЮЧЕНИЕ

Заключение

- Эффективно использовать современные процессоры невозможно без параллельного программирования
- Поддержка параллелизма в C++ пока лишь на базовом уровне: потоки, синхронизация и т.п.
  - Тема ||| активно развивается в C++ Standard Committee
- Для эффективной и продуктивной разработки применяйте готовые решения
  - Библиотеки параллельных шаблонов
  - Эффективные языковые расширения (если приемлемо)
  - Специализированные библиотеки с поддержкой |||

Успехов в параллельном программировании!

Спасибо за внимание!

Общие замечания к работам (1)

- Так делать не надо:

```
#include <iostream> // and lots of other headers
using namespace std;
int main() { ... }
```

- Множество имён добавляется в глобальное пространство, с риском конфликтов
- Хороший стиль – локализовать нужные имена:

```
#include <iostream>
int main() {
  using std::cout; using std::endl; // etc.
  cout << "Hello world!“ << endl;
}
```

- Также: `typedef`, приставка `std::`

Общие замечания к работам (2)

- Так делать не надо:

```
int main() {
  double * v1 = new double[N];
  double * v2 = new double[N];
  double result = scalar_product(v1,v2,N);
  std::cout << result;
  return 0;
} // Memory is not released!
```

- Вырабатывайте **привычку** освобождать полученные ресурсы!
- Ещё лучше: используйте `shared_ptr` (то есть RAII)

Общие замечания к работам (3)

```
bool done = false;
void Producer() {
  // fills the work queue here
  std::unique_lock<std::mutex> lock(a_mutex);
  done = true;
  condvar.notify_all();
}
void Consumer() {
  while(!done||has_work()) {
    std::unique_lock<std::mutex> lock(a_mutex);
    condvar.wait(lock, []{ return has_work()||done; });
    while (has_work()) do_work();
  }
}
```

- Это «гонка данных»!
- Решение: `std::atomic<bool> done;`