Platforma .NET – Wykład 15
Wybrane elementy
programowania równoległego w C#

Osoba prowadząca wykład, laboratorium i projekt:
dr hab. inż. Marek Sawerwain, prof. UZ

Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski

e-mail : M.Sawerwain@issi.uz.zgora.pl
tel. (praca) : 68 328 2321,
pok. 328a A-2,
ul. Prof. Z.Szafrana 2,
65-246 Zielona Góra

Ostatnia kompilacja pliku: Monday 5th June, 2023, t: 23:39

Notatki

Spis treści

Notatki

Plan wykładu – spotkania tydzień po tygodniu

(1) Informacje o wykładzie, pojęcie platformy, podstawowe informacje o platformie .NET
(2) Składowe platformy .NET: CLR, CTS, języki programowania, biblioteki klas, pojęcie podzespołu (ang. assembly)
(3) Programowanie w C# – środowisko VS, MonoDevelop, syntaktyka C#, wyjątki, współpraca z DLL
(4) Programowanie w C# – model obiektowy, typy uogólnione, lambda wyrażenia
(5) Programowanie w C# – aplikacje „okienkowe", programowanie wielowątkowe
(6) Programowanie w F# – podstawy, przetwarzanie danych tekstowych,
(*) "Klasówka I", czyli egzamin cześć pierwsza
(7) Dostęp do baz danych

Notatki

Plan wykładu – tydzień po tygodniu

(8) Język zapytań LINQ, Entity Framework
(9) Obsługa standardu XML
(10) Technologia ASP.NET 1/2
(11) Technologia ASP.NET 2/2
(12) Model widok i kontroler – Model View Controller
(13) Tworzenie usług sieciowych SOAP i WCF (komunikacja sieciowa)
(14) Wykład monograficzny .NET 1
(15) Wykład monograficzny .NET 2
(*) "Klasówka II", czyli egzamin cześć druga

Notatki

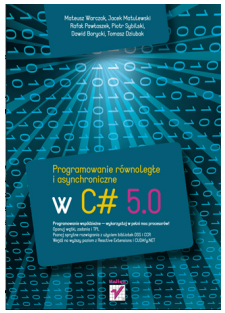
Plan wykładu

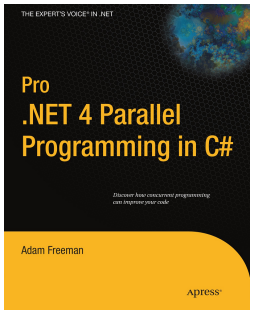
1. Wątki jako podstawowy składnik aplikacji równoległych
   - 1.1 podstawowa obsługa
   - 1.2 sekcje krytyczne, zdarzenia synchronizujące
   - 1.3 Monte Carlo na wątkach
2. Zadania zamiast wątków
   - 2.1 podstawy obsługi wątków,
   - 2.2 tworzenie i rola zadań,
   - 2.3 przykłady zadań,
   - 2.4
3. CUDA w .Net'cie
   - 3.1 technologia GPU
   - 3.2 siatka obliczeniowa
   - 3.3 pakiet CUDAfy
   - 3.4 przykładowe aplikacje

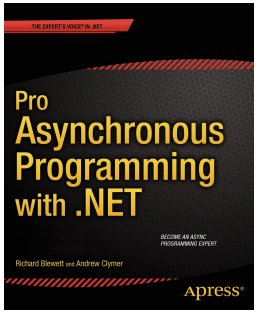
Notatki

Książki o programowaniu równoległym w .NET/C#

Od lewej: Mateusz Warczak, Jacek Matulewski, Rafał Pawłaszek, Piotr Sybilski, Dawid Borycki, Tomasz Dziubak, *Programowanie równoległe i asynchroniczne w C# 5.0*, Helion 2013., następnie: Adam Freeman, *Pro .NET 4 Parallel Programming in C#*, Apress 2010, pierwsza z prawej: Richard Blewett, Andres Clymer, Pro Asynchronous Programming with .NET, Apress, 2013







Notatki

Podstawowa obsługa wątków

Notatki

Czym jest wątek?

Obecne rozwiązania sprzętowo/programowe to środowiska wielowątkowe/wieloprocesorowe i tworzenie dla tego typu środowisk programów, to obecnie bardzo ważny obszar ze względu na rozwiązania wieloprocesorowe jakie stały się obecne niemal we wszystkich zastosowaniach począwszy od zagadnień naukowych, komercyjnych, a kończąc na rozwiązaniach domowych.

Czym jest wątek

Podstawowym elementem w systemie operacyjnym jest proces, realizujący określone zadanie. W ramach procesu mniejszą jednostką jest wątek, którego przeznaczeniem jest realizacja określonego zadania w ramach procesu. Każdy wątek zawiera obsługę wyjątków, podlega systemowi priorytetów, posiada także mechanizmy do zatrzymania realizacji zadania bądź jego wznowieniu.

Uwaga

Wątki obecnie na platformie .NET nie należy odnosić do systemu wątków danego systemu operacyjnego, w ramach którego działa maszyna CLR.

Notatki

Otoczenie klasy Thread – 1/2

Wątek to podstawowy element do tworzenia aplikacji równoległych (jednakże obecnie .NET 4.0/4.5/4.6 podstawowy obiekt do tworzenie aplikacji równoległych to zadanie – Task).

Podstawowa przestrzeń zwierająca typy do tworzenia aplikacji wielowątkowych to System.Threading. Zawarte są tam następujące typy:

- Interlocked – This type provides atomic operations for variables that are shared by multiple threads. Monitor This type provides the synchronization of threading objects using locks and wait/signals. The C# lock keyword makes use of a Monitor object under the hood.
- Mutex – This synchronization primitive can be used for synchronization between application domain boundaries.
- ParameterizedThreadStart – This delegate allows a thread to call methods that take any number of arguments.
- Semaphore – This type allows you to limit the number of threads that can access a resource, or a particular type of resource, concurrently.
- Thread – This type represents a thread that executes within the CLR. Using this type, you are able to spawn additional threads in the originating AppDomain.

Notatki

Otoczenie klasy Thread – 2/2

Wątek to podstawowy element do tworzenia aplikacji równoległych (jednakże obecnie .NET 4.0/4.5/4.6 podstawowy obiekt do tworzenie aplikacji równoległych to zadanie – Task).

Podstawowa przestrzeń zwierająca typy do tworzenia aplikacji wielowątkowych to System.Threading. Zawarte są tam następujące typy:

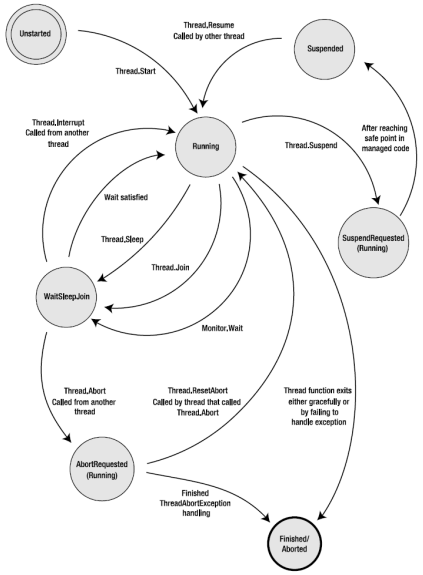
- ThreadPool – This type allows you to interact with the CLR-maintained thread pool within a given process.
- ThreadPriority – This enum represents a thread's priority level (Highest, Normal, etc.).
- ThreadStart – This delegate is used to specify the method to call for a given thread. Unlike the ParameterizedThreadStart delegate, targets of ThreadStart must always have the same prototype.
- ThreadState – This enum specifies the valid states a thread may take (Running, Aborted, etc.).
- Timer – This type provides a mechanism for executing a method at specified intervals.
- TimerCallback – This delegate type is used in conjunction with Timer types

Notatki

Stany wątku

Stany jakie wątek może przyjąć:

1. Unstarted
2. Running
3. SyspendedRequested,
4. Suspended
5. WaitSleepJoin
6. AbortRequested
7. Finised/Aborted



Notatki

Krótko o obsłudze wątków

Uruchomiemie wątku:

```
Thread t = new Thread( metodaDlaWatku );
t.Start();
```

Uśpienie wątku:

```
t.Sleep( m ) ; // m -- milisekundy
```

Wznowienie wątku:

```
t.Suspend();
```

Wątek tła:

```
t.IsBackground = true;
```

Wątek tła zostanie przerwany jeśli wątek główny aplikacji zakończy swoje działane. Wątki pierwszoplanowe w odróżnieniu od wątków tla zostanie dokończone nawet jeśli wątek główny zakończył swoje działanie.

Zmiana priorytetu wątku:

```
t.Priority = Highest;
```

Dopuszczalne wartości: Highest, AboveNormal, Normal, BelowNormal, Lowest.

Notatki

Krótko o obsłudze wątków

Wątek można przerwać za pomocą metody Abort. Choć nie jest to polecana metoda, wymaga przechwycenia wyjątku, można odwołać przerwanie wątku za pomocą ResetAbort. Większym problemem jest możliwość niedokończenia operacji atomowych.

```
static void metodaWatku() {
 try {
  // obliczenia
 }
 catch(ThreadAbortException e) {
  ...
 }
 catch(Exception e) {
  ...
 }
}
```

Oczekiwanie na zakończenia wątku:

```
t.Join();
```


Notatki

Sekcja krytyczna – lock

Utworzenie sekcji krytycznej realizuje się za pomocą słowa lock oraz referencji do jakiegokolwiek obiektu:

```
private Object thisLock = new Object();

lock (thisLock) {

  // operacje na zmiennej

}
```

Należy jednak pamiętać, że lock jest zamienina na obiekt monitora:

```
System.Threading.Monitor.Enter( thisLock );
try {
  // operacje na zmiennej
}
finally {
    System.Threading.Monitor.Exit(thisLock );
}
```


Notatki

Zdarzenia synchronizujące

Istnieją dwa typy zdarzeń synchornizujących **AutoResetEvent** oraz **ManualResetEvent**. Pierwszy typ samodzielnie zmienia swój stan na „nie aktywne zdarzenia" po uruchomieniu wątku. Drugi typ dostarcza kilku metod jak min. Set, Reset do sterowania sygnalizacją zdarzenia.

```
using System;
using System.Threading;

class ThreadingExample {
 static AutoResetEvent autoEvent;

 static void DoWork() {
  Console.WriteLine("   worker thread started, now waiting on event...");
  autoEvent.WaitOne();
  Console.WriteLine("   worker thread reactivated, now exiting...");
 }
 ...
```


Notatki

Zdarzenia synchronizujące

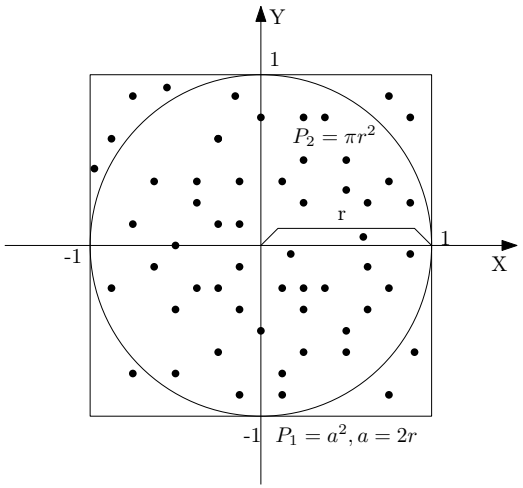
Istnieją dwa typy zdarzeń synchornizujących **AutoResetEvent** oraz **ManualResetEvent**. Pierwszy typ samodzielnie zmienia swój stan na „nie aktywne zdarzenia" po uruchomieniu wątku. Drugi typ dostarcza kilku metod jak min. Set, Reset do sterowania sygnalizacją zdarzenia.

```
...
 static void Main() {
  autoEvent = new AutoResetEvent(false);

  Console.WriteLine("main thread starting worker thread...");
  Thread t = new Thread(DoWork);
  t.Start();

  Console.WriteLine("main thread sleeping for 1 second...");
  Thread.Sleep(1000);

  Console.WriteLine("main thread signaling worker thread...");
  autoEvent.Set();
 }
}
```


Notatki

Liczba pi i Monte Carlo

Idea metody Monte Carlo przybliżająca wartość Pi sprowadza się do tego, że należy losować po dwa punkty x oraz y. Dla każdego wylosowanego punktu sprawdza się, czy mieści się on wewnątrz okręgu, tj. warunek $x^2 + y^2 < 1$. Wszystkie punkty, które znajdują się wewnątrz okręgu należy naturalnie zliczyć. Ponieważ stosunek pola okręgu do pola kwadratu wynosi $\pi/4$, to łatwo podać wzór, który posłuży nam do obliczenia liczby π, a będzie to 4 * cnt / mc. Wielkość cnt to liczba punktów o współrzęnych (x, y), znajdujących się wewnątrz okręgu, natomiast wartość *mc* to całkowita liczba wylosowanych punktów.



Notatki

MC dla liczby pi – szeregowo 1/2

Obliczenia MC dla liczby π w wersji szeregowej tj. obliczamy punkt po punkcie.

```
using System;
...
using System.Threading;

namespace Watki {
    class Program {

        static Random r = new Random();
        static void Main(string[] args) {
            uruchamianieObliczenPi();
        }

        static double obliczPi(long iloscProb) {
            double x, y;
            long iloscTrafien = 0L;

            for (long i = 0; i < iloscProb; i++) {
                x = r.NextDouble();
                y = r.NextDouble();
                if (x * x + y * y < 1) ++iloscTrafien;
            }
            return 4.0 * iloscTrafien / iloscProb;
        }
```


Notatki

MC dla liczby pi – szeregowo 2/2

Obliczenia MC dla liczby π w wersji szeregowej tj. obliczamy punkt po punkcie.

```
        static void uruchamianieObliczenPi()
        {
            int czasPoczatkowy = Environment.TickCount;

            long iloscProb = 10000000L;
            double pi = obliczPi(iloscProb);
            Console.WriteLine("Pi={0}, blad={1}", pi, Math.Abs(Math.PI - pi));

            int czasKoncowy = Environment.TickCount;
            int roznica = czasKoncowy - czasPoczatkowy;
            Console.WriteLine("Czas obliczen: " + (roznica).ToString());
        }

    }
}
```


Notatki

Liczba pi równolegle

Wersja obliczająca przybliżenie liczby Pi na wątkach:

```
using System;
...
using System.Threading;

namespace Watki {
    class Program {

        static Random r = new Random();
        const int ileWatkow = 8;
        static double pi = 0;
        private static Object LockForPi = new Object();

        static void Main(string[] args) {
            int czasPoczatkowy = Environment.TickCount;

            Thread[] tt = new Thread[ileWatkow];

            for (int i = 0; i < ileWatkow ; i++ ) {
                tt[i] = new Thread(uruchamianieObliczenPi);
                tt[i].Priority = ThreadPriority.Lowest;
                tt[i].Start();
            }
```


Notatki

Liczba pi równolegle

Wersja obliczająca przybliżenie liczby Pi na wątkach:

```
    foreach(Thread t in tt) {
        t.Join();
    }

    pi /= ileWatkow;

    Console.WriteLine("Pi={0}, blad={1}", pi, Math.Abs(Math.PI - pi));
    int czasKoncowy = Environment.TickCount;
    int roznica = czasKoncowy - czasPoczatkowy;
    Console.WriteLine("Całkowity czas obliczen: " + (roznica).ToString());
}
```

Pozostałe fragmenty działają w ramach wątku, wykorzują sekcję krytyczną do aktualizacji pola **pi**.

Notatki

Liczba pi równolegle

Wersja obliczająca przybliżenie liczby Pi na wątkach:

```
static double obliczPi(long iloscProb) {
    Random r = new Random( Program.r.Next() & DateTime.Now.Millisecond );
    double x, y;
    long iloscTrafien = 0L;

    for (long i = 0; i < iloscProb; i++)
    {
        x = r.NextDouble();
        y = r.NextDouble();
        if (x * x + y * y < 1) ++iloscTrafien;
    }
    return 4.0 * iloscTrafien / iloscProb;
}
```


Notatki

Liczba pi równolegle

Wersja obliczające przybliżenie liczby Pi na wątkach:

```
        static void uruchamianieObliczenPi()  {
            int czasPoczatkowy = Environment.TickCount;

            long iloscProb = 10000000L / ileWatkow;
            double pi = obliczPi(iloscProb);

            lock (LockForPi) { Program.pi += pi; }

            Console.WriteLine("Pi={0}, blad={1}, watek nr {2}",
                       pi, Math.Abs(Math.PI - pi),
                              Thread.CurrentThread.ManagedThreadId);

            int czasKoncowy = Environment.TickCount;
            int roznica = czasKoncowy - czasPoczatkowy;
            Console.WriteLine("Czas obliczen {0} dla watku {1} ",
                         (roznica).ToString(),
                                  Thread.CurrentThread.ManagedThreadId);
        }

    }

}
```


Notatki

Liczba pi równolegle – ale z pulą wątków

```
using System;
...
using System.Threading;

namespace Watki {

    class Program {

        static Random r = new Random();
        const int ileWatkow = 100;
        static double pi = 0;

        private static Object LockForPi = new Object();

        static void Main(string[] args) {
            int czasPoczatkowy = Environment.TickCount;

            WaitCallback metodaWatku = uruchamianieObliczenPi;
            ThreadPool.SetMaxThreads(30, 100);


            for (int i = 0; i < ileWatkow; i++) {
                ThreadPool.QueueUserWorkItem(metodaWatku, i);
            }
```


Notatki

Liczba pi równolegle – ale z pulą wątków

```
  int ileDostepnychWatkowWPuli = 0;
  int ileWszystkichWatkowWPuli = 0;
  int ileDzialajacychWatkowPuli = 0;
  int tmp = 0;
  do {
    ThreadPool.GetAvailableThreads(out ileDostepnychWatkowWPuli, out tmp);
    ThreadPool.GetMaxThreads(out ileWszystkichWatkowWPuli, out tmp);
    ileDzialajacychWatkowPuli = ileWszystkichWatkowWPuli
                                          - ileDostepnychWatkowWPuli;
    Console.WriteLine("Ilosc aktywnych watkow puli: {0}",
                                         ileDzialajacychWatkowPuli);
    Thread.Sleep(1000);
  } while (ileDzialajacychWatkowPuli > 0);

  pi /= ileWatkow;

  Console.WriteLine("Pi={0}, blad={1}", pi, Math.Abs(Math.PI - pi));

  int czasKoncowy = Environment.TickCount;
  int roznica = czasKoncowy - czasPoczatkowy;
  Console.WriteLine("Całkowity czas obliczen: " + (roznica).ToString());
}
```


Notatki

Liczba pi równolegle – ale z pulą wątków

```
static double obliczPi(long iloscProb)
{
    Random r = new Random(Program.r.Next() & DateTime.Now.Millisecond);
    double x, y;
    long iloscTrafien = 0L;

    for (long i = 0; i < iloscProb; i++)
    {
        x = r.NextDouble();
        y = r.NextDouble();
        if (x * x + y * y < 1) ++iloscTrafien;
    }
    return 4.0 * iloscTrafien / iloscProb;
}
```


Notatki

Liczba pi równolegle – ale z pulą wątków

```
        static void uruchamianieObliczenPi(object param)
        {
            int czasPoczatkowy = Environment.TickCount;
            int? indeks = param as int?;

            Console.WriteLine("Uruchamianie obliczen, watek nr {0}, indeks {1}",
                   Thread.CurrentThread.ManagedThreadId,
                   indeks.HasValue ? indeks.Value.ToString() : "---");

            long iloscProb = 10000000L / ileWatkow;
            double pi = obliczPi(iloscProb);

            lock (LockForPi) { Program.pi += pi; }

            Console.WriteLine("Pi={0}, blad={1}, watek nr {2}", pi,
                     Math.Abs(Math.PI - pi), Thread.CurrentThread.ManagedThreadId);

            int czasKoncowy = Environment.TickCount;
            int roznica = czasKoncowy - czasPoczatkowy;
            Console.WriteLine("Czas obliczen {0} dla watku {1} ",
                 (roznica).ToString(), Thread.CurrentThread.ManagedThreadId);
        }

    }

}
```


Notatki

Zadanie

Zadanie (ang. Task) to główne pojęcie w programowaniu równoległym w ramach C#. Pojęcie to zostało wprowadzone w .NET 4.0. I obecnie jest to podstawowa klasa w programowaniu równoległym w .NET'cie.

- zdanie/Task to wzorzec programowania asynchronicznego – Task-based Asynchronous Pattern (TAP),
- zdanie to typ **System.Threading.Tasks.Task** lub **System.Threading.Tasks.Task<TResult>**,
- łatwo przekazać dane do zdanie oraz odczytać rezultat,
- a także nakazać wykonanie kolejncyh czynności tuż po zakończeniu zadania,
- synchronizacja w dostępie do zmiennych odbywa się w sposób podobny do wątków,
- należy pamiętać iż zadania to pojęcie przykrywające wątki.

Notatki

Podstawowe zadanie

Podstawowe przykład uruchamiający dodatkowe zadanie w oddzielnym wątku z wykorzystaniem lambda wyrażenia.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1 {
    class Program {
        static void Main(string[] args) {
            Task.Factory.StartNew( () => {
                Console.WriteLine("Hej, hej jestem zadaniem!!!");
            });

            Console.WriteLine("Metoda main zakończona,
                                naciśniej [enter] aby zakończyć.");
            Console.ReadLine();
        }
    }
}
```


Notatki

Cztery sposoby tworzenia zadań

Jeśli mamy taką metodę:

```
static void printMessage() {
 Console.WriteLine("Hello World");
}
```

Zastosowanie delegatu Action:

```
Task task1 = new Task(new Action(printMessage));
```

Anonimowy delegat:

```
Task task2 = new Task(delegate { printMessage(); });
```

Wyrażenie lambda:

```
Task task3 = new Task(() => printMessage());
```

albo

```
Task task4 = new Task(() => { printMessage(); });
```

Uruchomienie zadań:

```
task1.Start();  task2.Start();  task3.Start();
task4.Start();
```


Notatki

Zadanie i parametry

Utworzenie zadań i przekazywanie danych do poszczególnych zadań:

```
string[] messages = { "First task", "Second task",
  "Third task", "Fourth task" };
```

Uruchomienie zadań z przekazaniem parametru:

```
foreach (string msg in messages) {
 Task myTask = new Task(obj => printMessage((string)obj), msg);
 myTask.Start();
}
```

Metoda printMessage tym razem ma następującą postać:

```
static void printMessage(string message) {
 Console.WriteLine("Message: {0}", message);
}
```


Notatki

Rezultat zadania

Utworzenie zadania które daje w wyniki wartość int:

```
Task<int> task1 = Task.Factory.StartNew<int>(() => {
 int sum = 0;
 for (int i = 0; i < 100; i++) {
  sum += i;
 }
 return sum;
});
```

Odczyt rezultatu zadania jest następujące:

```
Console.WriteLine("Result 1: {0}", task1.Result);
```


Notatki

Notatki

Zadanie po wykonaniu zadania

```
Task t1, t2, t3, t4;

t1 = new Task(() => { Thread.Sleep(1000);
   Console.WriteLine("zadanie t1 zakonczone id={0}", Task.CurrentId); });
t2 = new Task(() => { Thread.Sleep(2000);
   Console.WriteLine("zadanie t2 zakonczone id={0}", Task.CurrentId); });
t3 = new Task(() => { Thread.Sleep(3000);
   Console.WriteLine("zadanie t3 zakonczone id={0}", Task.CurrentId); });
t4 = new Task(() => { Thread.Sleep(4000);
   Console.WriteLine("zadanie t4 zakonczone id={0}", Task.CurrentId); });

Task[] zadania = { t1, t2, t3, t4 };

t2.ContinueWith((t) => {
 Console.WriteLine("zadanie o id {1} został wykonane po zakonczeniu zdania t2 o id {0}", t.Id, Task.CurrentId);
});

foreach (Task t in zadania) t.Start();
foreach (Task t in zadania) t.Wait();
```


Notatki

Dodatkowe opcje dla zadania

Dodatkowe opcje do zadania można przekazać w następujący sposób:

```
Task t = new Task(() => { ... }, TaskCreationOptions.None );
```

Dostepne opcje:

- None – Uses the default task creation options,
- PreferFairness – A request to the task scheduler to schedule tasks as fairly as possible,
- LongRunning – Specifies that the task will be long running, which is a hint to the task scheduler,
- AttachedToParent – Specifies that a child task is attached to a parent in the task hierarchy,
- DenyChildAttach – nie podłączyć zadania do zadań potomnych, próba podłącznie skutkuje wystąpieniem wyjątku,
- HideScheduler – wykorzystanie domyślnego planistę, nawet jeśli podano nowego planistę.

Producenci i konsumenci na zdaniach

Typowy problem programowania równoległe, producencie tworzą zasoby które muszą być przetworzone z zachowaniem spójności danych:

```
class BankAccount {
    public int Balance {
        get;
        set;
    }
}

class Deposit {
    public int Amount {
        get;
        set;
    }
}
```

Dostepna jest też kolekacja blokujące z opisem depozytów;

- BlockingCollection<Deposit> blockingCollection = new BlockingCollection<Deposit>();

Notatki

Producenci jako zadania

Utworzenie zadan tworzących depozyty.

```
Task[] producers = new Task[3];
for (int i = 0; i < 3; i++) {
  producers[i] = Task.Factory.StartNew(() => {
  for (int j = 0; j < 20; j++) {
    Deposit deposit = new Deposit { Amount = 100 };
    blockingCollection.Add(deposit);
  }
 });
};
```

Sygnał, że zakończono tworzenie depozytów:

```
Task.Factory.ContinueWhenAll(producers, antecedents => {
  Console.WriteLine("Signalling production end");
  blockingCollection.CompleteAdding();
});
```


Notatki

Kosument z kontem bankowym

Tworzenie obiektu konta, gdzie znajdzie się suma z depozytów:

```
BankAccount account = new BankAccount();
```

Zadanie dla konsumenta:

```
Task consumer = Task.Factory.StartNew(() => {
  while (!blockingCollection.IsCompleted) {
    Deposit deposit;
    if (blockingCollection.TryTake(out deposit)) {
      account.Balance += deposit.Amount;
    }
  }
  Console.WriteLine("Final Balance: {0}", account.Balance);
});
```

Oczekiwanie na zakończenie operacji konsumenta:

```
consumer.Wait();
```


Notatki

CUDA w .NET'cie

Notatki

Technologia GPU

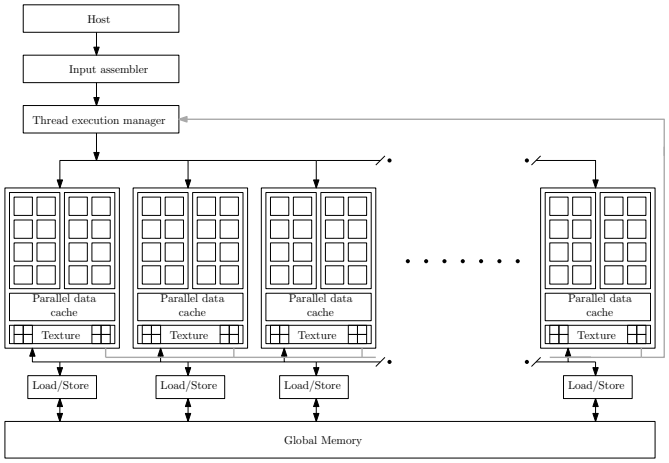
W największym skrócie, skoro kary graficzne są takie szybkie, to może zamiast grafiki warto liczyć inne zadania. Kilka zalet technologi GPU:

1. obliczenia uniwersalne nie tylko związane z grafiką komputerową,
2. znaczące przyspieszenie, zazwyczaj 10-krotne, 100-krotne, a są zadania gdzie można osiągnąć 1000-krotne przyspieszenie,
3. stosunkowa tania,
4. niższe zużycie energii w stosunku do wydajności,
5. jest stosowane wszędzie, Netbooki, laptopy, a nawet telefony komórkowe,
6. „względnie" łatwe programowanie, wsparcie producentów, np.: C++ AMP Microsoftu.

Notatki

Technologia GPU – CUDA

Ogólna architektura współczesnego procesora graficznego:

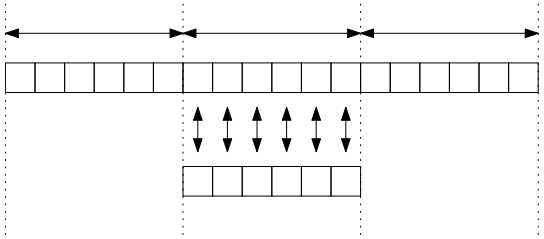


Notatki

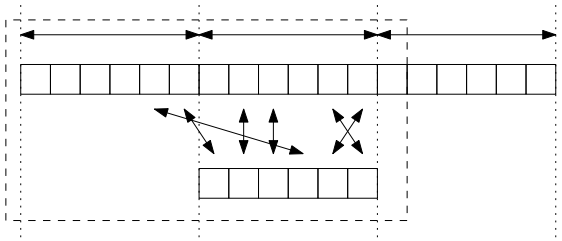
Technologia GPU – CUDA – Adresowanie danych

Uporządkowany dostęp do pamięci globalnej GPU, jest niezbędny dla osiągnięcia wysokiej wydajności:

szerokość okna dopasowana do wielkości pamięci L2

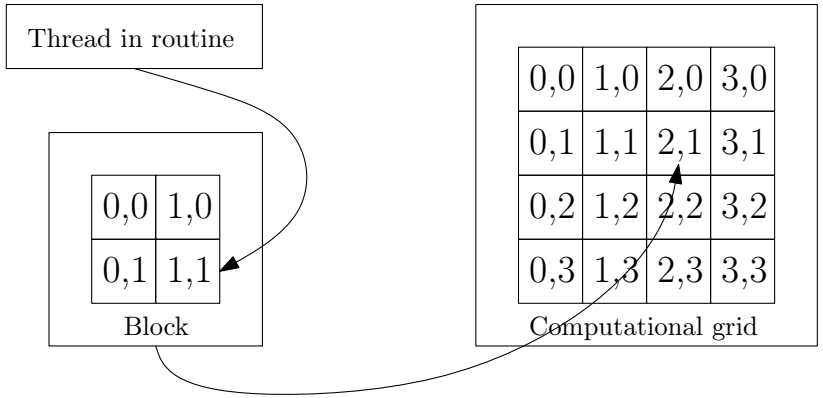


brak dopasowana ale wysoka wydajność dla serii kart 4xx i 5xx



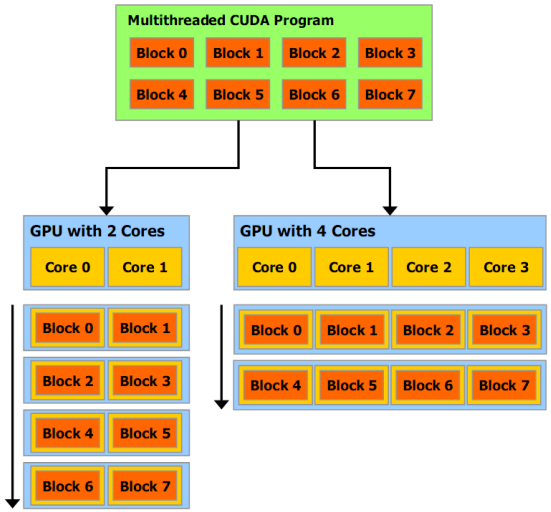
Notatki

Technologia GPU – CUDA – Siatka obliczeniowa



Notatki

Skalowalność architektury GPU



Notatki

Istniejące rozwiązania wspomagające obliczenia

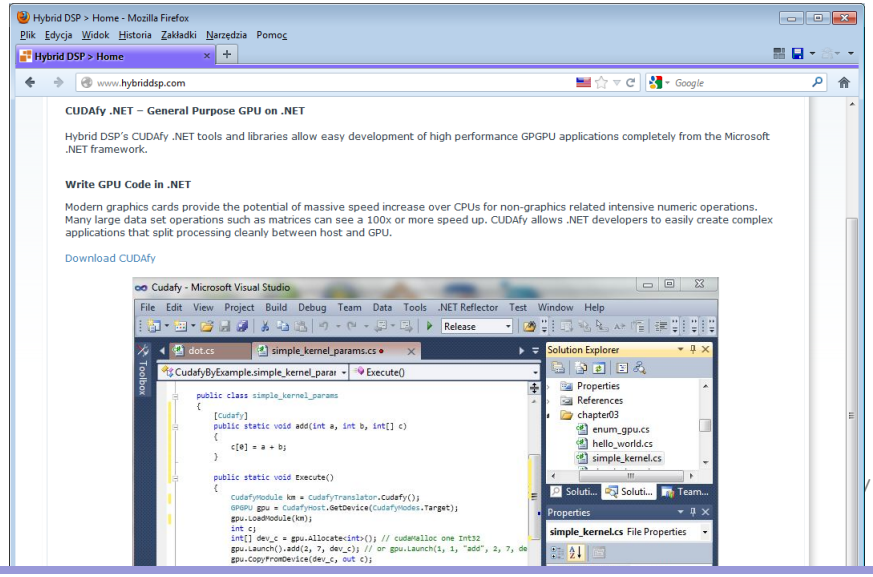
Warto wspomnieć o innych rozwiązaniach:

1. OpenCL – `http://www.khronos.org/opencl/`
2. Aparapi – `https://github.com/Syncleus/aparapi`
3. ArrayFire – `http://www.accelereyes.com/products/arrayfire`
4. Thrust – `https://thrust.github.io/`
5. C++ AMP – `http://msdn.microsoft.com/en-us/library/hh265137.aspx`
6. TornadoVM – `https://www.tornadovm.org/`
7. Taichi Lang – `https://www.taichi-lang.org/`
8. Intel One API – `https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html`

Notatki

CUDAfy .NET

Rozwiązanie opracowane przez HybridDsp, dwie wersje licencjonowania, GNU GPL oraz komercyjna (jedno stanowisko około 150 Euro, obecnie jednak dostępna bez dodatkowy kosztów – `https://github.com/lepoco/CUDAfy.NET`).



/ 62

Zalety CUDAfy .NET

1. Tworzenie oprogramowania w ramach .NET
2. Zgodność z modelem programowania CUDA
   - 2.1 pewne elementy poprawione (współpraca ze wskaźnikami),
   - 2.2 atrybut [Cudafy] oznaczający kod GPU,
   - 2.3 parametr GThread
3. Obsługa emulacji wykonania jądra obliczeniowego
4. Wsparcie dla różnych rodzajów pamięci dla GPU oraz hosta
5. Współpraca z typami .NET
6. Operacje asynchroniczne
7. wsparcie dla CUFFT, CUBLAS
8. Metody, stałe, struktury oraz stała pamięć,
9. „Mądre" tablice (np.: GetLength)

Notatki

Notatki

Cudafy – co trzeba wiedzieć

Najważniejsze pojęcia w programowaniu Cudafy:

1. atrybut [Cudafy]
2. klasa GThread (siatka obliczeniowa)
3. translator CudafyTranslator (tłumaczenie C# na CUDA C++),

Typowy scenariusz:

1. wczytanie modułu Cudafy w pliku XML,
2. utworzenie obiektu reprezentującego GPU,
3. załadowanie modułu do GPU,
4. inicjalizacja danych hosta,
5. alokacja danych w GPU,
6. przesłanie danych do GPU,
7. uruchomienie kernela,
8. odczyt danych z GPU,
9. koniec i albo powracamy do początku (1), albo wracamy do punktu (4).

Notatki

Czy CUDA jest na pokładzie?

Początki są łatwe:

(I) tworzymy zwykły projekt dla konsoli,
(II) dodajemy referencję do podzespołu Cudafy.NET (biblioteka DLL),
(III) upewniamy się, że mamy CUDA SDK,
(IV) i kartę ze stajni NVIDIA ;-).

```
using Cudafy;
using Cudafy.Host;
...
foreach (GPGPUProperties prop
      in CudafyHost.GetDeviceProperties(CudafyModes.Target))
{
    Console.WriteLine("   ---- Informacje o {0} ---", i);
    Console.WriteLine("Nazwa: {0}", prop.Name);
    Console.WriteLine("Id urzadzenia: {0}", prop.DeviceId);
    Console.WriteLine("Zdolnosc obliczeniowa:  {0}.{1}",
        prop.Capability.Major, prop.Capability.Minor);
    ...
}
```


Notatki

Prosty „kernel" na początek

Prosty kernel obliczeniowy do kompilacji za pomocą nvcc:

```
#include <stdio.h>

__global__ void add( int a, int b, int *c ) {
    *c = a + b;
}
int main( void ) {
    int c;
    int *dev_c;
    cudaMalloc((void**)&dev_c, sizeof(int)) ;

    add<<<1,1>>>( 2, 7, dev_c );

    cudaMemcpy(&c, dev_c,sizeof(int), cudaMemcpyDeviceToHost);

        printf("2 + 7 = %d\n", c);

        cudaFree(dev_c);

        return 0;
}
```


Notatki

Prosty „kernel" na początek

Prosty kernel obliczeniowy w .NETcie:

```
class Program {
    [Cudafy]
    public static void add(float a, float b, float[] c) {
        c[0] = a + b;
    }
    static void execute_test() {
        CudafyModule km = CudafyTranslator.Cudafy();
        GPGPU gpu = CudafyHost.GetDevice(CudafyModes.Target);
        gpu.LoadModule(km);
        float c = 0;
        float[] dev_c = gpu.Allocate<float>();
        gpu.Launch(1, 1, "add", 3.0f, 2.0f, dev_c);
        //or: gpu.Launch().add(3.0f, 2.0f, dev_c);
        gpu.CopyFromDevice(dev_c, out c);
        Console.WriteLine("3 + 2 = {0}", c);
        gpu.Free(dev_c);
    }

    static void Main(string[] args) {
        execute_test();
    }
```


Notatki

Dodawanie wektorów i siatka obliczeniowa

Niech A,B,C będą wektorami, zadanie jest następujące:

$$C = A + B$$

Przygotowanie danych:

```
int[] a = new int[N];
int[] b = new int[N];
int[] c = new int[N];

int[] dev_a = gpu.Allocate<int>(a);
int[] dev_b = gpu.Allocate<int>(b);
int[] dev_c = gpu.Allocate<int>(c);

gpu.CopyToDevice(a, dev_a);
gpu.CopyToDevice(b, dev_b);
```


Notatki

Przykłady – „coś" z wektorem

Uruchomienie procedury obliczeniowej i odczyt danych:

```
gpu.Launch(32, 1, addVectorsFaster, dev_a, dev_b, dev_c);
gpu.CopyFromDevice(dev_c, c);
```

Procedury obliczeniowa wolniejsza:

```
[Cudafy]
public static void addVectorsSlower(GThread thread,
        int[] a, int[] b, int[] c) {
    int tid = thread.blockIdx.x;
    if (tid < N) {
        c[tid] = a[tid] + b[tid];

    }
}
```


Notatki

Przykłady – „coś" z wektorem

Uruchomienie procedury obliczeniowej i odczyt danych:

```
gpu.Launch(32, 1, addVectorsFaster, dev_a, dev_b, dev_c);
gpu.CopyFromDevice(dev_c, c);
```

Procedury obliczeniowa i szybsza:

```
[Cudafy]
public static void addVectorsFaster(GThread thread,
        int[] a, int[] b, int[] c) {
    int tid = thread.blockIdx.x;
    while (tid < N) {
        c[tid] = a[tid] + b[tid];
        tid += thread.gridDim.x;
    }
}
```


Notatki

Szybkość – Liczby pseudolosowe

Funkcje generujące liczby losowe, podejście oparte o algorytm Marsagli MWC (ang. multiply with carry):

```
void default_seed() {
    m_w = 521288629;
    m_z = 362436069;
}

uint getuint() {
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

Będziemy generować $N = 1 << 20$ liczb wg. powyższego algorytmu i mierzyć czas.

Notatki

Szybkość – Liczby pseudolosowe

Funkcja generująca liczby pseudolosowe, pętla z rozwinięciem, wersja dla CPU:

```
void rnd_test_4_cpu() {
    int i;
    uint r1,r2,r3,r4;
    for (i = 0; i < N; i+=4) {
        r1 = getuint(); r2 = getuint();
        r3 = getuint(); r4 = getuint();
        rndTab[i + 0] = r1; rndTab[i + 1] = r1;
        rndTab[i + 2] = r1; rndTab[i + 3] = r1;
    }
}
```


Notatki

Szybkość – Liczby pseudolosowe

Funkcja mierząca czas wykonania:

```
static long profile_in_ms(int iterations, Action func) {
    GC.Collect();
    GC.WaitForPendingFinalizers();
    GC.Collect();

    func();

    var watch = Stopwatch.StartNew();
    for (int i = 0; i < iterations; i++) {
        func();
    }

    watch.Stop();

    return watch.ElapsedMilliseconds;
}
```


Notatki

Szybkość – Liczby pseudolosowe

Kernel generujący liczby pseudolosowe:

```
[Cudafy]
public static void rnd_generation_gpu(GThread thread,
        uint[] seed_m_z, uint[] seed_m_w, uint[] rnd_out_tab) {
    uint[] _sh_m_z =
      thread.AllocateShared<uint>("_sh_m_z", threadsPerBlock);
    uint[] _sh_m_w =
      thread.AllocateShared<uint>("_sh_m_w", threadsPerBlock);

    int tid = thread.threadIdx.x
                    + thread.blockIdx.x * thread.blockDim.x;
    int n = thread.threadIdx.x;

    if (tid == 0) {
        int i;
        for (i = 0; i < threadsPerBlock; i++ ) {
            _sh_m_z[i] = seed_m_z[i];
            _sh_m_w[i] = seed_m_w[i];
                }
    }
    thread.SyncThreads();
```


Notatki

Szybkość – Liczby pseudolosowe

Kernel generujący liczby pseudolosowe:

```
    while (tid < N) {
        _sh_m_z[n] = 36969 * (_sh_m_z[n] & 65535)
             + (_sh_m_z[n] >> 16);
        _sh_m_w[n] = 18000 * (_sh_m_w[n] & 65535)
             + (_sh_m_w[n] >> 16);

        rnd_out_tab[tid] = (_sh_m_z[n] << 16) + _sh_m_w[n];

        tid += thread.blockDim.x * thread.gridDim.x;
    }
}
```

Czas na pomiar:

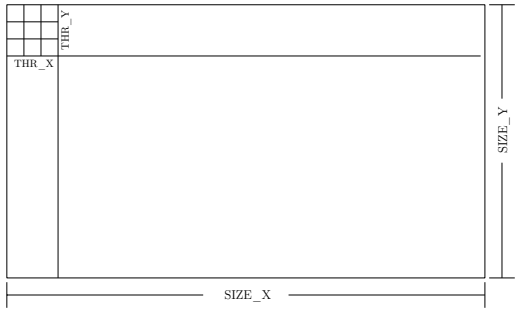
```
ms = profile_in_ms(100, rnd_test_4_cpu);
ms = profile_in_ms(1000, rnd_test_1_gpu);
```


Notatki

Rysowanie zbiorów fraktalnych

Wywołanie funkcji:

```
dim3 _blocks = new dim3(SIZE_X / 16, SIZE_Y / 16);
dim3 _threads = new dim3(16, 16);
gpu.Launch(_blocks, _threads).mandelbrot_draw_faster(
                    dev_image_ptr, scale, moveX, moveY);
gpu.CopyFromDevice(dev_image_ptr, image_ptr);
```



Notatki

Rysowanie zbiorów fraktalnych

Rysowanie obrazu:

```
[Cudafy]
public static void julia_draw(GThread thread, byte[] ptr,
        float scale, float moveX, float moveY) {
    int x = thread.blockIdx.x;
    int y = thread.blockIdx.y;
    int offset = x + y * thread.gridDim.x;

    int juliaValue = julia(x, y, scale, moveX, moveY);

    ptr[offset * 4 + 0] = (byte)(juliaValue);
    ptr[offset * 4 + 1] = (byte)(juliaValue);
    ptr[offset * 4 + 2] = (byte)(juliaValue);
    ptr[offset * 4 + 3] = 255;
}
```


Notatki

Rysowanie zbiorów fraktalnych

I funkcja, obliczająca wartość tzw. ucieczki:

```
[Cudafy]
public static int julia(int x, int y, float scale,
        float moveX, float moveY) {
    float cR = -0.8f; float cI = 0.156f ;
    float newR, newI, oldR, oldI; int count = 0;

    newR = 1.5f * ((float)x - SIZE_X / 2.0f)
        / (0.5f * scale * SIZE_X) + moveX;
    newI = ((float)y - SIZE_Y / 2.0f)
        / (0.5f * scale * SIZE_Y) + moveY;

    while((count <= 255) && ((newR*newR + newI*newI)<=4.0f)) {
        ++count;
        oldR = newR; oldI = newI;

        newR = oldR * oldR - oldI * oldI + cR;
        newI = 2 * oldR * oldI + cI;
    }
    return count;
}
```


Notatki

W następnym tygodniu między innymi

Wybrane pojęcia i zagadnienia omawiane na następnym wykładzie:

1. nie, to
2. już
3. ostatni
4. wykład.

Proponowane tematy prac pisemnych:

1. realizacja translacji z C# do języka stosowanego w GPU,
2. techniki optymalizacji kodu jąder obliczeniowych (computational kernels),
3. mini-test wydajności z rysowania zbiorów np. Mandelbrota i Juli.

Dziękuje za uwagę!!!

Notatki