**Języki Programowania, 13.01.2020**
**Zadanie na 5**

## Część I

Obliczanie skomplikowanych procesów fizycznych w często wielowymiarowych przestrzeniach wymaga dużej mocy obliczeniowej. Bardzo często pomocne jest tutaj zrównoleglenie problemu. Możemy na przykład naszą przestrzeń podzielić na siatkę n punktów i dokonać obliczenia w tym konkretnym punkcie. W ten sposób możemy stworzyć n wątków, które w każdym z tych punktów obliczą interesujacy nas proces. W naszym prostym przypadku policzymy sobie wartości dwuwymiarowej funkcji Gaussa w danym węźle sieci

Funkcja main:

```
int main()
{

  unique_ptr<Function2D> gauss(new Gauss2D(10,0,0,5,5));
  Lattice2D latt(15,-10,10,15,-10,10,gauss);
  latt.CalculateLattice();
  latt.Print();


  return 0;
}
```

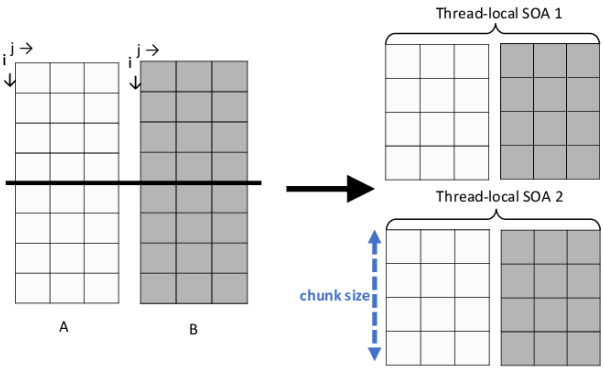
Piszemy klasę `Lattice2D`, która oblicza nam wartość funkcji zadanej funkcji (w naszym przypadku Gaussa) w punkcie. W pierwszej kolejności (część I), zadanie wykonujemy standardowo, po prostu implementując dwie zagnieżdżone pętle `for` po wierszach i kolumnach siatki.

## Część II

Uruchamianie wątków przynosi korzyści wtedy, gdy wątki te mogą rzeczywiście liczyć się równolegle. Gdy do dyspozycji mamy jeden komputer z jednym procesorem (a nie klaster komputerowy, gdzie jest wiele procesorów) jedyną możliwością jest wykorzystanie dostępnych rdzeni. Puszczananie większej ilości wątków po prostu nie ma sensu, gdyż one i tak będą się liczyć sekwencyjnie.

Chcemy zatem nasz program przystosować, aby dzielił on naszą dwuwymiarową tablicę na pewne przedziały, gdzie jeden wątek będzie liczył kilka węzłów należących do danego przedziału.

Ideę możemy sobie wyobrazić w następujący sposób:




Jak to zrobić w praktyce?

1. Dzielimy wątki tylko w jednej osi (x lub y)
2. Sprawdzamy, w której osi liczba węzłów jest większa (wybieramy x lub y w zależności od tego, która jest większą)
3. Dzielimy naszą tablicę na ilość części (podtablic) odpowiadającą ilości wątków.

4. W każdym wątki iterujemy po danej podtablicy i obliczamy wartości funkcji w punkcie

Jak sprawdzić ilość dostępnych rdzeni?
W terminalu: `grep -c ^processor /proc/cpuinfo`
W kodzie C++:

```
int concurentThreadsSupported = thread::hardware_concurrency();
```

Schemat postępowania:

1. Tworzymy klasę `Function2D` – klasa abstrakcyjna zawierająca funkcję `Eval` (do obliczania wartoścu funkcji w punkcie) oraz parametry wspólne dla wszystkich funkci 2D (np. zakres min, max, na osiach x oraz y)
2. Tworzymy klasę `Gauss2D` – klasa dziedziczy z `Function2D` i zawiera specyficzne pola dla dwuwymiarowej funkcji Gaussa oraz implementuje szczegółowo swoją funkcję `Eval`
3. Piszemy klasę `Lattice2D`, która posiada sprytny wskaźnik na obiekty klasy `Function2D` (dostoduje się do tego, że użyto `Gauss2D` w programie). W klasie tej implementujemy siatkę (definiujemy ilość węzłów) i wykonujemy obliczenia na każdym węźle używając funkcji `Eval` na sprytnym wskaźniku.