SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

NAPREDNE ARHITEKTURE RAČUNALA (220/250)

IZVJEŠTAJ 1

Toni Biuk

Tino Melvan

Split, travanj 2018.

Sadržaj

[1. SPECIFIKACIJE RAČUNALA 3](#_Toc511058905)

[2. Niti 4](#_Toc511058906)

[2.1. Kod 4](#_Toc511058907)

[2.1.1. Sekvencijalno 4](#_Toc511058908)

[2.1.2. Paralelno 5](#_Toc511058909)

[2.2. Rezultati 8](#_Toc511058910)

[3. Mutex 10](#_Toc511058911)

[3.1. Kod 10](#_Toc511058912)

[3.2. Rezultati 11](#_Toc511058913)

[PRILOG 12](#_Toc511058914)

# SPECIFIKACIJE RAČUNALA

U ovom poglavlju su navedene relevantne specifikacije računala korištenih za izvršavanje programa.

|  |  |
| --- | --- |
| OS | Microsoft Windows 10 Education Build 16299.309 |
| CPU | Intel® Core™ i7-7700HQ @ 2.80GHz |
| CPU # of Cores | 4 |
| CPU # of Threads | 8 |
| RAM | 16GB (2X8GB) DDR4 – 2400MHz |

Tablica 1: Specifikacije računala 1

|  |  |
| --- | --- |
| OS | Microsoft Windows 10 Pro Build 16299 |
| CPU | AMD FX-6300 @ 3.50GHz |
| CPU # of Cores | 3 |
| CPU # of Threads | 6 |
| RAM | 16 GB (8x2) DDR3 – 1066Mhz |

Tablica 2: Specifikacije računala 2

# Niti

## Kod

### Sekvencijalno

#include <Utility/Random.hpp>

#include <Utility/Traces.hpp>

#include <chrono>

#include <cstdint>

#include <cstdlib>

#define ARRAY\_LENGTH 100000

#define ARRAY\_MIN\_RANGE -1000

#define ARRAY\_MAX\_RANGE 1000

typedef std::chrono::high\_resolution\_clock Clock;

int32\_t main()

{

const Random random;

int32\_t array[ARRAY\_LENGTH];

auto result = 0;

for (auto& num : array)

{

num = random.Next(ARRAY\_MIN\_RANGE, ARRAY\_MAX\_RANGE);

DEBUG("[Main] Adding to array: %d\n", num);

}

const auto clockStart = Clock::now();

for (const auto& num : array)

{

result += num;

}

const auto clockEnd = Clock::now();

INFO("[Main] Execution time: %lld microsec\n",

std::chrono::duration\_cast<std::chrono::microseconds>

(clockEnd - clockStart).count());

INFO("[Main] Final result: %d\n", result);

return EXIT\_SUCCESS;

}

Kod 1: Program za sekvencijalno zbrajanje niza

### Paralelno

#include <Utility/Random.hpp>

#include <Utility/Traces.hpp>

#define HAVE\_STRUCT\_TIMESPEC

#include <pthread.h>

#include <chrono>

#include <cstdint>

#include <cstdlib>

#include <cmath>

#define ARRAY\_LENGTH 200000

#define ARRAY\_MIN\_RANGE -1000

#define ARRAY\_MAX\_RANGE 1000

#define NUMBER\_OF\_THREADS 4

struct ArgStruct {

int32\_t threadId;

int32\_t \*arrayStart;

int32\_t \*arrayEnd;

int32\_t result;

};

typedef std::chrono::high\_resolution\_clock Clock;

void\* GetPartialSum(void \*arguments)

{

auto \*args = static\_cast<struct ArgStruct \*>(arguments);

auto result = 0;

DEBUG("[Thread %d] Started...\n", args->threadId);

DEBUG("[Thread %d] Starting with %d\n", args->threadId, \*args->arrayStart);

DEBUG("[Thread %d] Ending with %d\n", args->threadId, \*args->arrayEnd);

while (args->arrayStart <= args->arrayEnd)

{

result += \*args->arrayStart;

args->arrayStart++;

}

args->result = result;

DEBUG("[Thread %d] Calculated %d\n", args->threadId, result);

return nullptr;

}

int32\_t main()

{

DEBUG("[Main] Starting program...\n");

DEBUG("[Main] ARRAY\_LENGTH: %d\n", ARRAY\_LENGTH);

DEBUG("[Main] NUMBER\_OF\_THREADS: %d\n", NUMBER\_OF\_THREADS);

const Random random;

struct ArgStruct args[NUMBER\_OF\_THREADS];

pthread\_t threads[NUMBER\_OF\_THREADS];

int32\_t array[ARRAY\_LENGTH];

auto result = 0;

const auto range = static\_cast<int>

(static\_cast<float>(ARRAY\_LENGTH) / NUMBER\_OF\_THREADS);

DEBUG("[Main] Division range: %d\n", range);

DEBUG("[Main] Making random array...\n");

for (auto& num : array)

{

num = random.Next(ARRAY\_MIN\_RANGE, ARRAY\_MAX\_RANGE);

DEBUG("[Main] Adding to array: %d\n", num);

}

const auto clockStart = Clock::now();

for (auto id = 0; id < NUMBER\_OF\_THREADS; id++)

{

args[id].threadId = id;

args[id].arrayStart = array + id \* range;

(id == NUMBER\_OF\_THREADS - 1) ? args[id].arrayEnd = &array[ARRAY\_LENGTH - 1]

: args[id].arrayEnd = array + (id + 1) \* range - 1;

const auto rc = pthread\_create(&threads[id],

nullptr,

GetPartialSum,

&args[id]);

if (rc)

{

INFO("[Main] ERROR; return code from pthread\_create() is %d\n", rc);

return EXIT\_FAILURE;

}

}

DEBUG("[Main] Joining threads!\n");

for (const auto& t : threads)

{

pthread\_join(t, nullptr);

}

DEBUG("[Main] Calculating result!\n");

for (const auto& arg : args)

{

result += arg.result;

}

const auto clockEnd = Clock::now();

INFO("[Main] Execution time: %lld microsec\n",

std::chrono::duration\_cast<std::chrono::microseconds>

(clockEnd - clockStart).count());

INFO("[Main] Final result: %d\n", result);

return EXIT\_SUCCESS;

}

Kod 2: Program za paralelno zbrajanje niza

## Rezultati

U poglavlju 2.1 se nalazi kod programa koji paralelno računa sumu niza koristeći niti. Kod 1 je primjer sekvencijalnog izvođenja programa, a Kod 2 paralelnog izvođenja programa. Veličina niza je definirana s ARRAY\_LENGTH, a broj niti s NUMBER\_OF\_THREADS. Program je paraleliziran na način da svaka nit izračuna sumu jednog dijela niza i zatim se rezultati niti zbroje u konačno rješenje.

Graf 1 prikazuje prosječno vrijeme izvršavanja programa. Vidi se da je sekvencijalno izvođenje programa znatno brže od paralelnog, te da vrijeme izvršavanja raste povećanjem broja niti. To je zato što u ovom primjeru, vrijeme izračuna sume je zanemarivo u usporedbi s vremenom kreiranja niti, pokretanja niti i prebacivanja konteksta tj. niti ne obavljaju dovoljno posla da bi bile isplative.

Graf 1: Brzina izvršavanja programa

Za bolje prikazivanje rada niti, program je izmijenjen dodavanjem koda za usporavanje funkcije (Kod 2). Vrijeme izvođenja izmijenjenog programa je prikazano na Grafu 2. Vidi se da kada vrijeme izvođenja funkcije nije trivijalno, vrijeme izvođenja programa se linearno smanjuje porastom broja niti ako je broj niti manji od 8. To je zato što procesor korištenog računala podržava istovremeno izvođenje 8 niti. Dodatnim povećanjem brojem niti brzina izvođenja ostaje ista.

Graf 2:Spora funkcija, Računalo 1

//GRAF I TEXT ZA DRUGI PC

# Mutex

## Kod

#include <Utility/Traces.hpp>

#include <mutex>

#include <thread>

#include <cstdint>

#include <vector>

#define NUM\_OF\_ITERATIONS 1000U

#define NUM\_OF\_THREADS 8U

void Increment(uint32\_t &n, std::mutex &mutex)

{

for (auto i = 0U; i < NUM\_OF\_ITERATIONS; ++i)

{

mutex.lock();

n++;

for (auto i = 0; i < 30; i++)

{

time(NULL);

}

mutex.unlock();

}

}

int32\_t main()

{

auto val = 0U;

std::vector<std::thread> threads;

std::mutex mutex;

for (auto i = 0U; i < NUM\_OF\_THREADS; i++)

{

threads.emplace\_back(std::thread(Increment,

std::ref(val),

std::ref(mutex)));

}

for (auto& thread : threads)

{

thread.join();

}

INFO("%u\n", val);

return EXIT\_SUCCESS;

}

Kod 3: Problem sinkronizacije

## Rezultati

//MUTEX, DATA RACE

# PRILOG

void\* GetPartialSum(void \*arguments)

{

auto \*args = static\_cast<struct ArgStruct \*>(arguments);

auto result = 0;

DEBUG("[Thread %d] Started...\n", args->threadId);

DEBUG("[Thread %d] Starting with %d\n", args->threadId, \*args->arrayStart);

DEBUG("[Thread %d] Ending with %d\n", args->threadId, \*args->arrayEnd);

while (args->arrayStart <= args->arrayEnd)

{

result += \*args->arrayStart;

args->arrayStart++;

for (auto i = 0; i < 300; i++)

{

time(NULL);

}

}

args->result = result;

DEBUG("[Thread %d] Calculated %d\n", args->threadId, result);

return nullptr;

}

Kod 4: Spora funkcija

#ifndef UTILITY\_RANDOM\_HPP

#define UTILITY\_RANDOM\_HPP

#include <cstdint>

#include <random>

class Random

{

public:

    Random();

    explicit Random(uint32\_t seed);

    int32\_t Next() const;

    int32\_t Next(int32\_t ceiling) const;

    int32\_t Next(int32\_t floor, int32\_t ceiling) const;

    double NextDouble() const;

    float NextFloat() const;

private:

    std::default\_random\_engine& globalURNG() const;

    void Randomize() const;

    void Randomize(uint32\_t seed) const;

};

#endif // !UTILITY\_RANDOM\_HPP

Kod 5: Random.hpp

#ifndef UTILITY\_TRACES\_HPP

#define UTILITY\_TRACES\_HPP

#include <cstdio>

#define INFO(fmt, ...) do { fprintf(stderr, fmt, \_\_VA\_ARGS\_\_); } while (0);

#if \_DEBUG

#define DEBUG(fmt, ...) do { INFO(fmt, \_\_VA\_ARGS\_\_); } while (0);

#else

#define DEBUG(mt, ...)

#endif

#endif // !UTILITY\_TRACES\_HPP

Kod 6: Traces.hpp