Tutorijal: TBB, II deo

Sadržaj

- Kontejneri
- Sinhronizacija

Konkurentni kontejneri

- Intel TBB obezbeđuje kontejnere koji se mogu bezbedno konkurentno koristiti
  - Konkurentne operacije nisu bezbedne nad STL (Standard Template Library) kontejnerima
  - Obično se STL kontejneri zaključavaju isključivim pristupom, što smanjuje paralelizam
- TBB kontejneri imaju lošije performanse od STL ako ih koristi jedna nit, ali imaju bolju skalabilnost
- Mogu da se koriste sa TBB-om, OpenMP-om ili običnim (pthread/c++11) nitima

Konkurentni red `concurrent_queue<T>`

- Zadržava lokalni FIFO poredak
  - Ako jedna nit stavi, a druga nit izvadi iz reda dve vrednosti, one izlaze istim redom kojim su stavljene. Ako više niti stavljaju i vade vrednosti konkurentno, FIFO poredak nije zagarantovan.
- Operacije za vađenje iz reda:
  - Neblokirajuća: `bool try_pop(T&)`
- Ugrađena podrška za iteriranje kroz red prilikom otklanjanja grešaka (debugging)

Konkurentni vektor

`concurrent_vector<T>`

- Dinamički proširivi niz tipa T
  - `grow_by(n)`
  - `grow_to_at_least_(n)`
- Elementi se ne pomeraju kada se vektor proširuje
- Moguć je konkurentni pristup i proširivanje
- Metode za brisanje i uništavanje vektora nisu bezbedne za konkurentno izvršavanje sa metodama za pristup ili proširivanje

Konkurentna mapa

`concurrent_hash_map<Key,T,HashCompare>`

- Asocijativna tabela koja preslikava ključ *Key* na element tipa *T*
- `HashCompare` je klasa koja određuje kako se ključevi prave i upoređuju
- Dozvoljava konkurentni pristup za čitanje i upis:
  - `bool insert(accessor &result, const Key &key)` za dodavanje ili izmenu,
  - `bool find(accessor &result, const Key &key)` za izmenu,
  - `bool find(const_accessor &result, const Key &key)` za pristup,
  - `bool erase(const Key &key)` za uklanjanje.

concurrent_hash_map

- Ponaša se kao kontejnter elemenata std::pair<const Key, T>
- Pristupom ovakvim elemntima se podrazumeva čitanje ili osvežavanje njihovih vrednosti
- Za čitanje se koristi klasa `const_accessor` a za osvežavanje običan `accessor`
- Samo jedan `accessor` može menjati element u jednom trenutku a više `const_accessor`-a može očitavati isti element u paraleli

concurrent_hash_map

- Objekat se oslobađa implicitno na kraju bloka u kom je akcesor definisan ili eksplicitno pozivom metode `accessor.release()`
- Od poziva `insert` ili `find` tj. prvog korišćenja pa sve do uništenja objekta, pristup će biti aktivan odnosno pristup mapi će biti zaključan
- Zato se savetuje oslobađanje ovog objekta što je pre moguće

concurrent_hash_map

- Struktura tipa `accessor` ima 2 polja:
  - `first` koje predstavlja ključ
  - `second` koje predstavlja vrednost
- Primer

```
StringTable::accessor a;
key: a->first
value: a->second
```

Primer 2: `concurrent_hash_map`

- Primer pravi konkurentnu mapu, gde su ključevi stringovi, a odgovarajući podaci predstavljaju broj pojavljivanja svakog od stringova u zadatom nizu.

```
typedef concurent_hash_map<string,int,MyHashCompare> StringTable;

// Function object for counting occurrences of strings
struct Tally {
  StringTable& table;
  Tally(StringTable& table_) : table(table_) {}
  void operator()(const blocked_range<string*> range) const {
    for(string* p=range.begin(); p!=range.end(); ++p) {
      StringTable::accessor a;
      table.insert(a, *p);
      a->second += 1;
    }
  }
};
```

Primer 2: `concurrent_hash_map`

- Struktura `HashCompare`:
  - Ima metodu `hash` koja definiše na koji način se obavlja “heširanje”; može biti nešto napredno kao u primeru ispod ali i nešto jednostavno poput prostog `return x;`
  - Ima metodu equal koja služi za poređenje elemenata

```
// Structure that defines hashing and comparison operations for user's type.
struct MyHashCompare {
    static size_t hash( const string& x ) {
        size_t h = 0;
        for( const char* s = x.c_str(); *s; ++s )
            h = (h*17)^*s;
        return h;
    }
    //! True if strings are equal
    static bool equal( const string& x, const string& y ) {
        return x==y;
    }
};
```

Sadržaj

- Kontejneri
- Sinhronizacija

Sinhronizacija

- TBB nudi dva mehanizma za međusobnu isključivost:
  - Muteksi: zaključavanje objekta pristupa, međusobna isključivost čitanja i pisanja.
  - Atomske operacije: zasnivaju se na atomskim operacijama koje nudi procesor. Jednostavnije su i brže od muteksa, ali su ograničene na uži skup tipova podataka.

Muteksi: `spin_mutex`

- Nit koja pokušava da zaključa već zaključan spin_mutex, mora da čeka da on bude otključan.
- spin_mutex je prigodan kada se njime zaključava nekoliko instrukcija.
- Obavezno je dodeliti ime objektu zaključavanja, u suprotnom C++ kompajler može prerano da ga ukloni.

Primer: spin_mutex

```
Node* Freelist;
typedef spin_mutex FreeListMutexType;
FreeListMutexType FreeListMutex;

Node* AllocateNode() {
  Node* n;
  {
    FreeListMutex::scoped_lock mylock(FreeListMutex);
    n = FreeList;
    if (n)
      FreeList = n->next;
  }
  if (!n)
    n = new Node();
  return n;
}
void FreeNode(Node* n) {
  FreeListMutexType::scoped_lock mylock(FreeListMutex);
  n->next = FreeList;
  FreeList = n;
}
```

Atomske operacije

- Druge niti vide atomske operacije kao trenutne.
- Primer:

Ako postoji samo jedna nit, može se napisati:

```
--x;
if (x==0) action();
```

Međutim, ako postoji više niti, operacije dva zadatka mogu da se učešljaju:

Task A	Task B
Ta = x x = Ta - 1 if (x==0)	Tb = x x = Tb - 1 if (x==0)

Atomske operacije (nastavak)

- Kako bi se rešio ovaj problem, mora se obezbediti da samo jedna nit radi dekrementiranje u bilo kom momentu, i da se provera vrednosti izvršava nad rezultatom dekrementa.

```
atomic<int> x;
if (--x==0) action();
```

- Metoda `atomic<int>::operator--` deluje automatski; ni jedna druga nit ne može da se umeša.

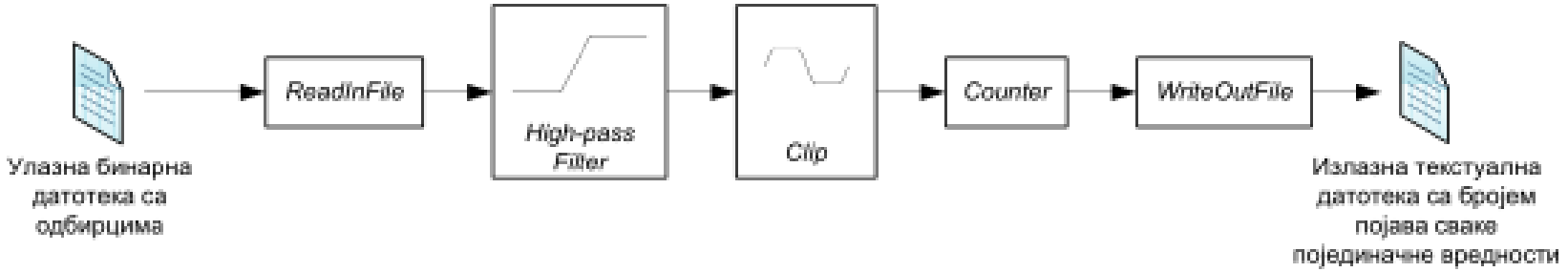
Dodatni materijali

- Samostalno analizirati priložene primere
- Knjiga Paralelno programiranje
- TBB dokumentacija: vodiči za programiranje i priručnici

Priprema za Zadatak 1 (1/2)

- `Extern` – govori da je promenljiva deklarisana u drugoj datoteci (izvan trenutnog opsega)
- Jedan od razloga upotrebe jeste da bi se neki podaci mogli koristiti u više datoteka bez nepotrebnog kopiranja
- U slučaju da povezivač ne pronađe datu promenljivu/funkciju koja je deklarisana kao ekstrena, pojaviće se Unresolved External Symbol greška

Priprema za Zadatak 1 (2/2)



Слика 1: Пример блокова обраде сигнала

- Na svakoj strelici se nalazi neka od struktura podataka koja je u “susednim” datotekama definisana kao eksterna
- Kada promenite u jednoj datoteci, morate promeniti u svim ostalim kako ne bi došlo do greške