

Multithreading C++11. 2-р хэсэг

F.CS306 ПАРАЛЛЕЛ ПРОГРАММЧЛАЛ – Лекц 8

Г.ГАНБАТ
ganbatg@must.edu.mn

Хичээлийн агуулга

- ❖ Статик тархалтанд суурилсан хуваарилалтууд
- ❖ Ачааллын тэнцвэрийг удирдах
- ❖ thread-үүд болон нөхцөл хувьсагчдийн харилцаа
- ❖ Thread Pool

СТАТИК ТАРХАЛТАНД СУУРИЛСАН ХУВААРИЛАЛТ

- ❖ CPU цөмийн тооноос хэтэрхий олон thread хувилах нь *oversubscription* үүсэх нөхцөлд хүргэдэг.
- ❖ CPU цөмийн тооноос олон байхад for-loops давталтын үед өгөгдлийг статикаар хуваарилална.
 - ❖ Блок тархалт
 - ❖ Цикл тархалт
 - ❖ Блок-цикл тархалт
- ❖ *lambdas* гэдэг нэргүй функцын capture механизмаар олон өгөгдлийг thread-руу зохистой дамжуулна.

```
[capture] (parameters) ->returnType {statement}
```

```
auto f = [](int a, int b) int { return a + b; };
auto n = f(1, 2);
```

БЛОК ТАРХАЛТ

- ❖ $m \% p \neq 0$ байх p ширхэг цөм, m ширхэг даалгавартай үед тус бүр нь m/p (хуваалтын хэмжээ – chunk size) даалгаврыг гүйцэтгэх p ширхэг thread хувилна.
- ❖ Хуваалтын хэмжээг hpc_helper.hpp толгой файлын *SDIV* макрог ашиглан тооцоолно.

$$SDIV(x, y) = \left\lfloor \frac{x+y-1}{y} \right\rfloor \geq \frac{x}{y}, \text{ энд } x, y \in \mathbb{R}^+$$

thread 0				thread 1				...	thread $p - 1$			
0	1	2	3	4	5	6	7	...	$m - 4$	$m - 3$	$m - 2$	$m - 1$

Цикл тархалт

- ❖ с ширхэг даалгаварыг р алхамаар thread бүрт Round-robin зарчимаар хуваарилна.
 - ❖ task id, task id+p, task id+2p, ...

threads	0	1	2	...	$p - 1$	0	1	2	...	$p - 1$...
tasks	0	1	2	...	$p - 1$	p	$p + 1$	$p + 2$...	$2p - 1$...

БЛОК-ЦИКЛ ТАРХАЛТ

- ❖ m ширхэг даалгавруудын c урттай блокыг p ширхэг thread бүрт дараалан ажиллуулахаар хуваарилдаг.
 - ❖ $m > s$ байхад thread бүрт $s = p * c$ алхамтайгаар давтана.
 - ❖ Хэрэв $c = 1$ бол цикл тархалт, $c = m/p$ бол блок тархалт

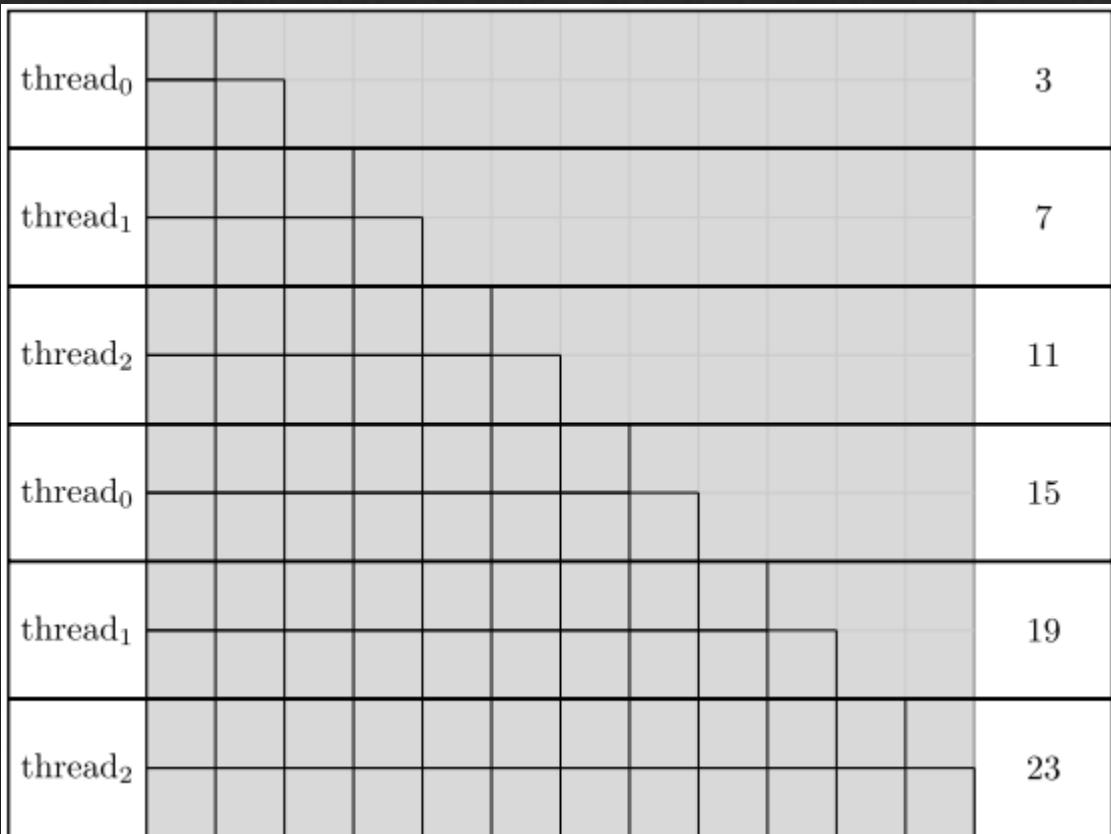
thread 0	thread 1	...	thread $p - 1$	thread 0	thread 1	...					
0	1	2	3	...	$s - 2$	$s - 1$	s	$s + 1$	$s + 2$	$s + 3$...

АЧААЛЛЫН ТЭНЦВЭРИЙГ УДИРДАХ

- ❖ *load imbalance*: Даалгавруудын зарим нь их цаг зарцуулдаг бол бусад даалгаврууд дуусчихаад байхад тухайн даалгавар ажилласаар байна.
 - ❖ Тэнцэрийг хадгалахын тулд thread-ийн **статик** болон **динамик** тархалтын үед авч үзнэ.
- ❖ Жишээ: 65000 ширхэг 28×28 хэмжээтэй 0 – 9 цифрийн гараар бичсэн хар цагаан зураг өгөгдсөн. Хос цифр бүрийн зайн матрицыг тооцоолно.
 - ❖ Цифр бүрийг $n = 28 * 28$ ширхэг утгатай векторт хөрвүүнэ.
 - ❖ $m = 65000$ зургуудыг $m \times n$ хэмжээтэй $D_{ij} = x_j^{(i)}$ матрицад хадгална. Энд m нь зургийн дугаар, j нь зургийн цэгийн дугаар
 - ❖ $O(m^2 * n)$ хугацаанд $m \times m$ хэмжээтэй зайн матрицыг байгуулна. Энд матрицын диагоналиас доош тооцоолоход хангалттай

Статик тархалтын тэнцвэр

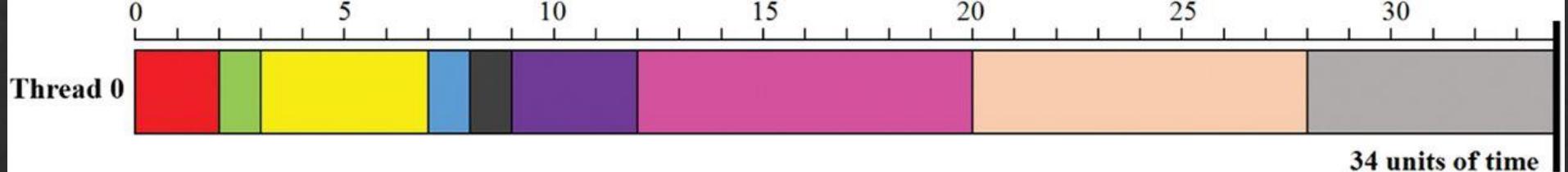
- ❖ Хуваалтын хэмжээ том байхад блок болон блок-цикл тархалт зэрэг статик хуваарь нь тохиромжгүй.



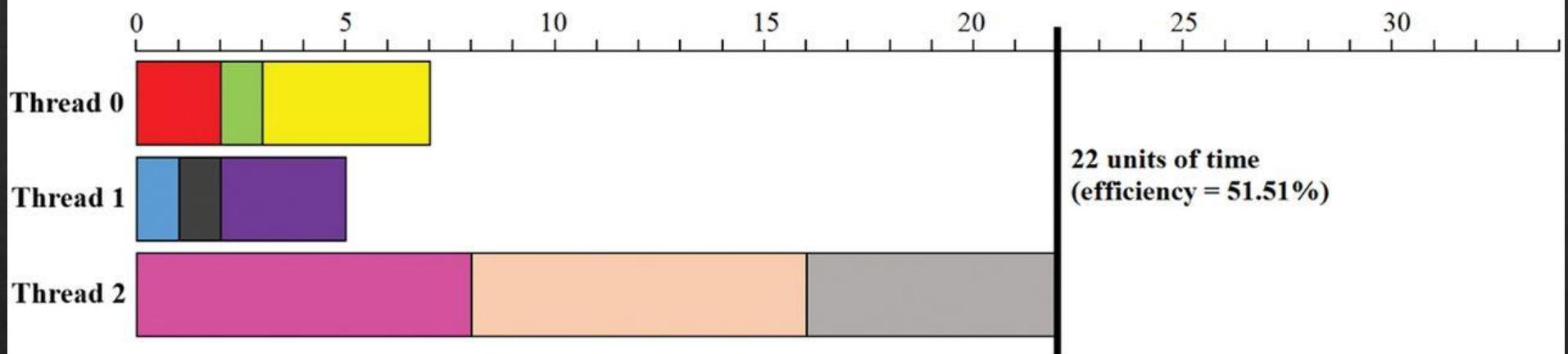
- ❖ 12x12 хэмжээтэй хос мэдээллийн зайн матриц дээр ажиллаж байгаа 2 хуваалтын хэмжээ бүхий 3 thread-ийн блок-циклийн тархалт.
- ❖ Баруун талд байгаа тоонууд нь нэг хуваалт дах зайн тооцоолын хэмжээг харуулсан.

Динамик блок-цикл тархалт

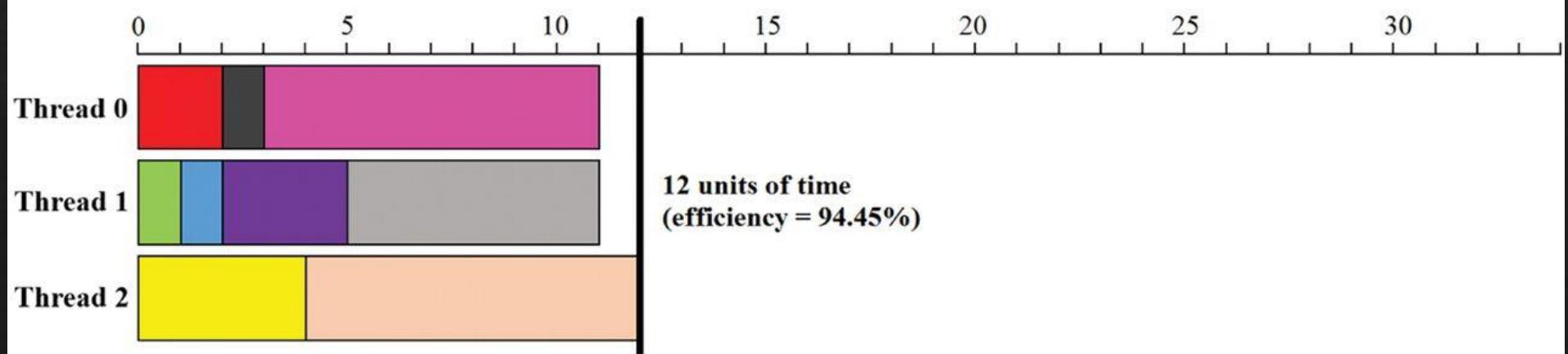
- ❖ Ажиллагааны үеийн даалгавар хуваарилалтыг **динамик хуваарилалт** гэнэ. **Даалгавар** болон **Хуваалтын түвшинд** сууринна. Эхнийх нь дараагийн хуваарилалтын хуваалтын хэмжээ $c = 1$ байх тохиолдол
 - ❖ Жижиг хуваалтын хэмжээ нь өсөлттэй ажлын хуваарилалтад тохиромжтой.
- ❖ Ажиллагааны явцад бүх мөрийн дуустал хуваалтуудыг динамикаар сонгож тооцооллол хийнэ.
 - ❖ Thread бүрийн тооцоолсон хуваалт хэмжээг глобал хувьсагчид цуглуулах ёстой учир энэ хувьсагч дээр Race condition үүснэ.
 - ❖ Үр дүнг зөв цуглуулахын тулд дээрх хандалтыг бие биендээ зөвшөөрөх хэрэгтэй.
- ❖ **mutex:** C ++ 11 нь эгзэгтэй хэсгийн гүйцэтгэлд thread-үүд харилцан бие биенээ хязгаарлах механизм



(a) Sequential



(b) Parallel (static scheduling)



(c) Parallel (dynamic scheduling)

Туршилт

- ❖ Xeon E5-2683 v4 CPU (2×16 физик цөм + hyperthreading)
- ❖ 64 битийн программ хангамжийн thread-үүд
- ❖ Хуваалтын хэмжээ c дээрх хурдсалт болон ажиллах хугацааг авч үзсэн

Дүгнэлт: Бүх хуваалтын хэмжээний хувьд хуваалтуудын динамик хуваарилалт ашигтай байна. Түүнчлэн ачаалал ихтэй даалгавруудыг боловсруулахад жижиг хуваалтын хэмжээ тохиромжтой байгаа нь харагдаж байна.

Mode	Chunk size c	1	4	16	64	256	1024
Static	Time in s	44.6	45.0	45.6	49.9	57.0	78.5
	Speedup	40.5	40.0	39.5	36.1	31.6	22.9
Dynamic	Time in s	43.6	43.6	43.9	46.3	53.8	77.6
	Speedup	41.3	41.3	41.0	38.9	33.5	23.2

Thread-үүд болон нөхцөлт хувьсагчдийн харилцаа

- ❖ Онолын хувьд (race-to-sleep) зогссон thread ямар ч энерги зарцуулагдахгүй ч заримдаа thread хоорондын хамаарлаас шалтгаалж thread-үүдийг унтуулахад шаардагдана.
 - ❖ Ямар нэг даалгавар ажлаа дуусгахыг хүлээн тооцооллоо хийж чадахгүй байгаа харилцан хамааралтай даалгавруудын гинж.
 - ❖ Динамик хуваариар thread-үүд глобал тоолуурыг нэмэгдэхийг хүлээдэг ч mutex-ийг хүлээхдээ бүх thread завгүй байдаг
- ❖ Бусад thread-үүдийг хүлээхээс өөр юу ч хийхгүй байгаа thread дээр тооцооллын нөөцийг гарздах нь тохиромжгүй.
 - ❖ **Нөхцөлт хувьсагчид:** Thread-үүдийг унтуулаад (sleep) дараа нь тэдэнд сэргээх (wake-up) дохио хялбар өгөх механизм.

Нөхцөлт хувьсагч

- ❖ Thread үргэлжлүүлэн ажиллахаасаа өмнө **нөхцөл**(*condition*) үнэн эсэхийг шалгаж болно. Нөхцөл үнэн болохыг хүлээхийн тулд thread нь **нөхцөлт хувьсагч** (*condition variable*) хэрэглэнэ.
 - ❖ Дохио өгч байвал thread ажиллахад бэлэн гэсэн үг
 - ❖ Яг ажиллах цаг нь болсон эсэхийг өөр төлвөөр (өөр дундын хувьсагч) тодорхойлно
- ❖ Нөхцөлт хувьсагч нь гүйцэтгэлийн зарим төлөв (зарим нөхцөл) биелээгүй байхад (нөхцлийг хүлээхэд) thread-үүд өөрсдийгөө байрлуулах дараалал юм;
 - ❖ Өөр thread төлвийг өөрчлөх үед эдгээр хүлээгдэж (*waiting*) байгаа thread-ийн нэгийг (эсвэл хэд хэдийг) сэргээх ба улмаар үргэлжүүлэн ажиллахыг зөвшөөрнө (**нөхцөлд дохио өгснөөр-signaling**).

Signaling thread

1. Signaling thread нь mutex-гийг *mutex.lock()* эсвэл *std::lock_guard*, *std::unique_lock* зэргийг хэрэглэн олж авах ёстой.
2. Lock-г эзэмшиж байх үед дундын төлөв өөрчлөгдеж, дараагийн дараалсан ажлыг гүйцэтгэдэг (командын шугамд хэвлэх гэх мэт).
3. Lock-г шууд *mutex.unlock()* хэрэглэж чөлөөлнө. Эсвэл *std::lock_guard*, *std::unique_lock* зэрэг нь дууссанаар суллагдана.
4. Нөхцөлт хувьсагчийн *notify_one()* функцээр нэг thread-д, *notify_all()* функцээр бүх thread-д дохиог илгээнэ.

Waiting threads

1. Хүлээж буй thread ижил mutex хэрэглэн *std::unique_lock*-ийг signal-ын шатанд олж авна. *std::lock_guard* энд хэрэглэгдэхгүй.
2. Түгжигдсэн үедээ *wait_until()* эсвэл нөхцөлт хувьсагчийн *wait()*, *wait_for()* функцийг дуудна. Бусад thread-үүд нь mutex-ийг дахин эзэмших боломжтой болгохын тулд түгжээ нь автоматаар тайлагддаг.
3. Нөхцөлт хувьсагч мэдэгдэх, нөхцөлт хувьсагчийн *wait()*, *wait_for()* функц ажиллаж дуусах зэрэг тохиолдолд thread сэрэх ба түгжээг олж авна. Энэ үед үргэлжлүүлэх эсвэл дахин унтах эсэхийг тодруулахын тулд дундын глобал төлвийг шалгана.

ONE-SHOT синхрончлол

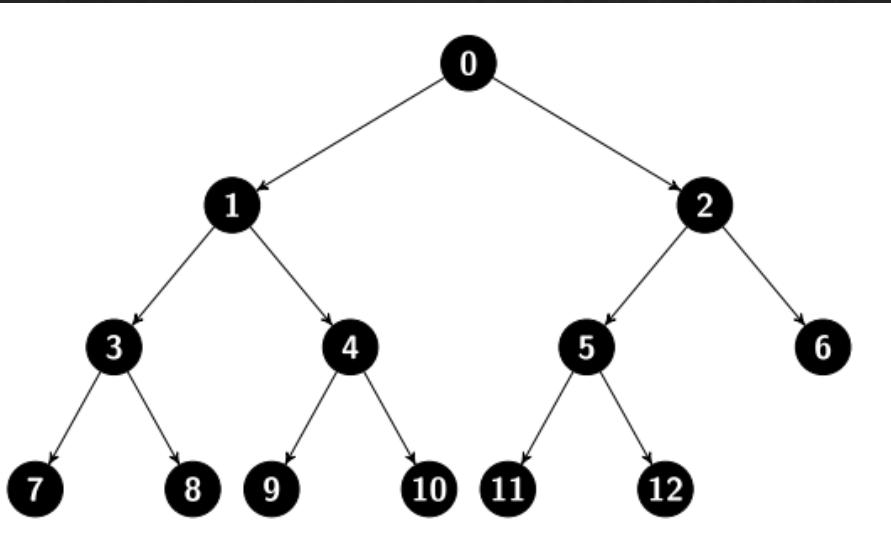
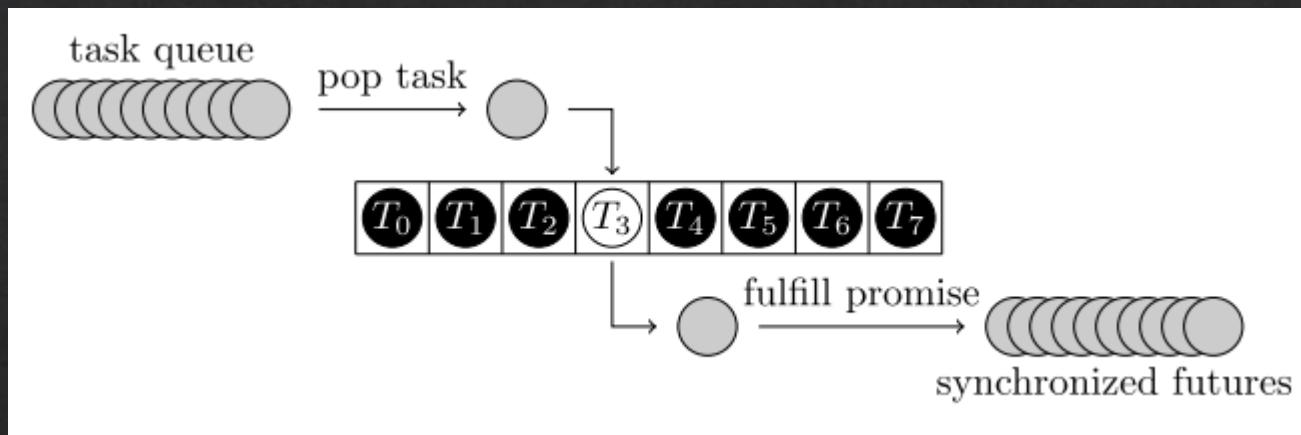
- ❖ Давтагдах синхрочлолын хэлбэртэй эсвэл олон thread-ийн гүйцэтгэлтэй байхад нөхцөлт хувьсагч ашигтай. (ping-pong)
- ❖ Нэг эсвэл хэд хэдэн thread нэг л удаа дохио өгөх бол *Future*, *Promise* хэрэглэдэг *one-shot синхрончлол* тохиромжтой: *auto future = promise.get_future();*
- ❖ Нэгээс олон thread-ийн утга түгээхэд *shared futures* хэрэглэнэ: *auto shared_future = promise.get_future().share();*

Implicitly Enumerable Sets дээрх паралеллчал

- ❖ Зарим алгоритм тодорхойгүй тооны даалгавар бүхий шугаман бус топологиор ажилладаг.
- ❖ Жишээ: Directed acyclic graph(DAG), Boggle, Web server
 - ❖ RAM-д багтахгүй, дискнээс дараалуулан унших боломжгүй
 - ❖ Тохирсон математик илэрхийлэл мэдэгдэхгүй
- ❖ Жишээ: Web server
 - ❖ Хэзээ ч хамаагүй хэдэн ч хуудас дуудаж болно
 - ❖ Хариулах хугацаа нь хүсэлтээс хамаараад өөр өөр байна.
- ❖ Энгийн арга: Даалгавар бүрт шинэ thread хувилна.
Oversubscription үүснэ, Гүйцэтгэлийн үзүүлэлтэд хохиролтой,
Denial of Service (DOS) халдлагад өртөх
- ❖ *Thread pool:* Даалгавар тасралтгүй өсдөг тохиолдолд тогтмол тооны thread хэрэглэнэ.

Thread pool

Ирсэн даалгаврын хүсэлт үүдийг шугаман топологтой өгөгдлийн бүтцэд жагсаана. for-loop ашиглана.



Ирсэн даалгаврын хүсэлт үүдийг мод хэлбэрийн топологтой өгөгдлийн бүтцэд жагсаана. Preorder зарчмаар гүйцэт модны тойролт ашиглана.

Thread pool:
Шугам топологи
дээрх хэрэглээнийй
жишээ

```
#include <iostream>
#include "threadpool.hpp" // the to be written pool

ThreadPool TP(8);           // 8 threads in a pool

int main () {

    // function to be processed by threads
    auto square = [](const uint64_t x) {
        return x*x;
    };

    // more tasks than threads in the pool
    const uint64_t num_tasks = 32;
    std::vector<std::future<uint64_t>> futures;

    // enqueue the tasks in a linear fashion
    for (uint64_t task = 0; task < num_tasks; task++) {
        auto future = TP.enqueue(square, task);
        futures.emplace_back(std::move(future));
    }

    // wait for the results
    for (auto& future : futures)
        std::cout << future.get() << std::endl;
}
```

Баярлалаа.