

Superkompiuteriai. Lygiagretusis programavimas MPI

Karolis Ryselis

Kauno Technologijos Universitetas



Have you heard of the fastest Fugaku supercomputer? It can run an infinite loop in 4 seconds!

Paskaitos turinys

- 1 Klasteriai ir superkompiuteriai
- 2 MPI
- 3 MPI komunikavimas tarp procesų

Klasteriai ir superkompiuteriai

Mazgas *node*. Kompiuteris, turintis vieną ar daugiau skaičiavimo vienetų.

Klasteris *cluster*. Susijusių mazgų rinkinys.

Išteklių tinklas *grid*. Klasterių rinkinys.

Superkompiuteris *supercomputer*. Kompiuteris, savo paleidimo metu esantis vienas pirmaujančių pasaulyje pagal skaičiavimo galią.

Mazgas



Klasteris



Superkompiuterių pajėgumas¹



¹<https://www.top500.org/statistics/perfdevel/>

Superkompiuterių OS

- Top 500 superkompiuterių operacinių sistemų pasiskirstymas:
 - Linux — 500
- Linux yra efektyvi sistema aukštos spartos kompiuterių tinklui.
- Linux yra nemokama — Windows licencijos kainuotų daug pinigų.
- Jei kažkas neveikia Linux sistemoje, galima pataisyti pačiam.
- Perkrauti reikia labai retai.

Galingiausi superkompiuteriai

Nr.	Sistema	Šalis	EFlop/s	MW	Brand.	Metai
1	Frontier	JAV	1.19	22.7	8.70M	2022
2	Fugaku	Japonija	0.44	29.9	7,63M	2020
3	LUMI	Suomija	0.15	2.9	1.11M	2022
4	Leonardo	Italija	0.24	7,4	1,82M	2023
5	Summit	JAV	0.15	10	2,41M	2018

MPI

- MPI (*Message Passing Interface*) — pranešimų perdavimo funkcijų, skirtų realizuoti lygiagrečiuosius algoritmus, standartas.
- MPI realizacija — MPI priemonių biblioteka klasteriui, paskirstytos atminties superkompiuteriui ar heterogeniniam tinklui.
- MPI veikia principu SPMD (*Single Program Multiple Data*).
- Procesai yra programos kopijos ir sukuriami prieš programos vykdymą.
- Procesai neturi bendros atminties, bet vykdo tą patį kodą.
- Komunikacija tarp procesų vyksta siuntinėjant žinutes.

MPI

- MPI 1.0 išleista 1994 m.
- MPI 4.1.6 — paskutinė versija, išleista 2023 m. rugsėjį.
- MPI yra standartas, turintis keletą realizacijų, tiek komercinių, tiek nekomercinių.
- Modulyje bus naudojama OpenMPI.

OpenMPI

- OpenMPI yra atviro kodo projektas.
- Naudojamas daugelyje Top 500 superkompiuterių.
- Palaikoma įvairiose lygiagrečiųjų skaičiavimų aplinkose.

Pagrindinės MPI funkcijos

```
Init(int& argc, char **&argv)
```

Nurodo MPI darbo pradžią. Iki funkcijos iškvietimo negalima kviesti kitų MPI funkcijų.

```
Finalize()
```

Nurodo MPI darbo pabaigą. Po funkcijos iškvietimo negalima kviesti kitų MPI funkcijų.

```
int Get_rank()
```

Grąžina proceso numerį.

```
int Get_size()
```

Grąžina procesų kiekį.

MPI programos struktūra

```
#include <mpi.h>
//...
int main(int argc, char **argv){
    //...
    // Cannot call MPI functions here
    MPI_Init(&argc, &argv);
    // Can call MPI functions here
    MPI_Finalize();
    // Cannot call MPI functions here
    //...
}
```

OpenMPI komandos

- Programos kompiliavimas, kai programos tekstas yra faile `program.cpp`, sukurtą vykdomąjį failą norime pavadinti `program`:
`mpicxx program.cpp -o program`
- Programos `program` vykdymas, kai norime paleisti 5 procesus:
`mpirun -n 5 program`
- Jei reikia, galima vykdyti MPI programą keliuose kompiuteriuose prieš vykdymą sukūrus šių kompiuterių sąrašo failą:
`mpirun --hostfile hosts -n 50 program`

MPI tiesioginis ryšys (*point-to-point communication*)

- Komunikavimas vyksta tarp dviejų procesų.
- Siuntėjas siunčia žinutę gavėjui, gavėjas — priima.
- Komunikavimas vyksta komunikatoriaus viduje.
- Gavėjas identifikuojamas savo numeriu (*rank*) komunikatoriuje.

MPI tiesioginis ryšys — žinutės siuntimas

```
Comm::Send(const void* buf, int count, const Datatype&  
datatype, int dest, int tag)
```

Comm komunikatorius — procesų rinkinys, kuriame siunčiama žinutė.

buf siunčiamos žinutės pradžios adresas.

count siunčiamos žinutės dydis.

datatype siunčiamos informacijos tipas.

dest proceso, kuriam siunčiama, numeris.

tag žinutės skiriamoji žymė.

MPI tiesioginis ryšys — žinutės gavimas

```
Comm::Recv(const void* buf, int count, const Datatype&  
datatype, int tag, Status& status)
```

Comm komunikatorius — procesų rinkinys, kuriame siunčiama žinutė.

buf atminties, kurioje išsaugoma žinutė, adresas.

count atminties, kurioje išsaugoma žinutė, dydis.

datatype gaunamos informacijos tipas.

tag žinutės skiriamoji žymė.

status informacija apie gautą žinutę.

MPI komunikatoriai

Communicator abstrakti struktūra, nusakanti MPI procesų, kurie gali komunikuoti tarpusavyje, grupę.

MPI::COMM_WORLD visų galimų procesų rinkinys.

- Programuotojas gali susikurti savo komunikatorių, kurį sudaro `MPI::COMM_WORLD` poaibis.
- Proceso numeris apibrėžiamas komunikatoriaus ribose.
- Pranešimai siunčiami tarp pasirinkto komunikatoriaus procesų.

MPI duomenų tipai

- MPI::CHAR
- MPI::INT
- MPI::SIGNED_CHAR
- MPI::UNSIGNED_CHAR
- MPI::UNSIGNED_SHORT
- MPI::UNSIGNED_LONG
- MPI::LONG_DOUBLE
- MPI::PACKED
- MPI::COMPLEX
- MPI::LONG_DOUBLE_COMPLEX
- MPI::SHORT
- MPI::LONG
- MPI::UNSIGNED
- MPI::DOUBLE
- MPI::FLOAT
- MPI::BOOL
- MPI::BYTE
- MPI::WCHAR
- MPI::DOUBLE_COMPLEX

MPI proceso numeris

- Procesai numeruojami iš eilės nuo 0.
- Eilės numeris skirtas identifikuoti procesus `Send` ir `Recv` kreipiniuose.
- `Recv` gali priimti pranešimus tiek iš nurodyto proceso (pagal numerį), tiek iš bet kurio proceso (nurodoma `MPI::ANY_SOURCE`).

MPI žinutės žymė

- Žymė yra sveikasis skaičius, naudojamas nurodyti žinutės tipui.
- Recv priima pranešimą su nurodyta žyme. Jei reikia priimti bet kokią pranešimą, naudojama žymė `MPI::ANY_TAG`.

MPI žinutės buferis

- Recv keipinyje reikia nurodyti žinutės priėmimo buferį. Jis gali būti paprastas kintamasis arba masyvas.
- Jei žinutės ilgis mažesnis už buferio dydį, užpildoma dalis buferio.
- Jei buferis per mažas, gaunama perpildymo klaida.
- Probe funkcija galima patikrinti, koks priimamos žinutės dydis.

Comm::Probe

```
Probe(int source, int tag, Status& status)
```

source proceso, iš kurio norime priimti žinutę, numeris komunikatoriuje arba MPI::ANY_SOURCE

tag žymė, su kuria pažymėtą pranešimą norime priimti, arba MPI::ANY_TAG

status informacija apie gaunamą žinutę.

Pranešimų sinchronizavimas

- MPI standartas neapibrėžia sinchronizavimo pilnumo.
- Sinchronizavimas gali būti:
 - Pilnai sinchronizuotas — siuntėjas laukia, kol gavėjas gaus, gavėjas laukia, kol siuntėjas išsiųs.
 - Buferizuotas — siuntėjas laukia, kol žinutė bus įrašyta į buferį, gavėjas laukia, kol žinutė atsiras buferyje.

MPI apsikeitimas pranešimais

```

int main() {
    MPI::Init();
    auto rank = MPI::COMM_WORLD.Get_rank();
    auto totalProcesses = MPI::COMM_WORLD.Get_size();
    if (rank == 0) {
        cout << "Process count " << totalProcesses << endl;
        char name[MPI::MAX_PROCESSOR_NAME];
        int name_length = 0;
        MPI::Get_processor_name(name, name_length);
        cout << "Processor name " << name << endl;
        int sent_message = 0, received_message = 0;
        MPI::COMM_WORLD.Send(&sent_message, 1, MPI::INT, 1, 1);
        cout << "Sent message " << sent_message << endl;
        MPI::COMM_WORLD.Recv(&received_message, 1, MPI::INT, 1, 1);
        cout << "Received message " << received_message << endl;
    } else {
        int sent_message = 1, received_message = 1;
        MPI::COMM_WORLD.Recv(&received_message, 1, MPI::INT, 0, 1);
        cout << "Received message " << received_message << endl;
        MPI::COMM_WORLD.Send(&sent_message, 1, MPI::INT, 0, 1);
        cout << "Sent message " << sent_message << endl;
    }
    MPI::Finalize();
    return 0;
}

```

MPI apsikeitimas pranešimais

Programos paleidimas ir rezultatas

```
/usr/bin/mpirun -np 2 mpi_send_recv_1
```

```
Process count 2
```

```
Processor name ryselis-B450-AORUS-ELITE-V2
```

```
Sent message 0
```

```
Received message 1
```

```
Received message 0
```

```
Sent message 1
```

MPI priėmimas iš keleto procesų

```

int main() {
    Init();
    auto rank = COMM_WORLD.Get_rank();
    if (rank == 0) {
        auto size = COMM_WORLD.Get_size();
        for (int i = 1; i < size; i++) {
            Status status;
            COMM_WORLD.Probe(ANY_SOURCE, 1, status);
            char buffer[status.Get_count(CHAR)];
            COMM_WORLD.Recv(buffer, status.Get_count(CHAR), CHAR,
                            status.Get_source(), 1);
            string message(&buffer[0], &buffer[status.Get_count(CHAR)]);
            cout << "Received value " << message << endl;
        }
    } else {
        string message = get_message(rank);
        auto *buffer = message.c_str();
        COMM_WORLD.Send(buffer, (int) message.size(), CHAR, 0, 1);
    }
    Finalize();
}

```

MPI priėmimas iš keleto procesų

```
/usr/bin/mpirun -np 6 mpi_send_recv_2
```

```
Received value Two
```

```
Received value Four
```

```
Received value Five
```

```
Received value Three
```

```
Received value One
```

MPI suma

```

void calculate_full_sum() {
    auto worker_count = COMM_WORLD.Get_size() - 1;
    const auto DATA_SIZE = 13000;
    int numbers[DATA_SIZE];
    // fill numbers array with data
    auto chunk_size = DATA_SIZE / worker_count;
    for (auto i = 1; i <= worker_count; i++) {
        int end_index = (i == worker_count ? DATA_SIZE: (i+1)*chunk_size);
        int start_index = i * chunk_size;
        int current_chunk_size = end_index - start_index;
        COMM_WORLD.Send(numbers+start_index, current_chunk_size, INT, i,
            TAG_PARTIAL_ARRAY);
    }
    for (auto i = 0; i < worker_count; i++) {
        COMM_WORLD.Recv(&partial_sums[i], 1, INT, ANY_SOURCE,
            TAG_PARTIAL_SUM);
    }
    auto total_sum = accumulate(&partial_sums[0],
        &partial_sums[worker_count], 0,
        [](int x, int y) { return x + y;});
    cout << total_sum << endl;
}

```

MPI suma

```
void calculate_partial_sum() {  
    Status status;  
    COMM_WORLD.Probe(MAIN_PROCESS, TAG_PARTIAL_ARRAY, status);  
    const auto item_count = status.Get_count(INT);  
    int items[item_count];  
    COMM_WORLD.Recv(items, item_count, INT, status.Get_source(),  
                     status.Get_tag());  
    auto total_sum = accumulate(items, items + item_count, 0,  
                                [](int x, int y) { return x + y;});  
    COMM_WORLD.Send(&total_sum, 1, INT, 0, TAG_PARTIAL_SUM);  
}
```

MPI suma

```
int main() {  
    Init();  
    auto rank = COMM_WORLD.Get_rank();  
    if (rank == 0) {  
        calculate_full_sum();  
    } else {  
        calculate_partial_sum();  
    }  
    Finalize();  
    return 0;  
}
```