TRƯỜNG ĐẠI HỌC BÁCH KHOA, ĐHQG TP.HỒ CHÍ MINH

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



PARALLEL PROCESSING

Assignment

NHÂN MA TRẬN SỬ DỤNG MPI

|  |  |
| --- | --- |
| Tutor: | TS Thoại Nam |
| Class: L01, | Group: |
|  | Lê Thị Thanh Thảo 1713177 |
|  | Võ Đình Khương 1711835 |
|  | Nguyễn Linh Đăng Minh 1712177 |

Ho Chi Minh, 9/2019

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

Contents

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [1](#page3) | | | Giới thiệu | 1 |
| [2](#page3) | | | Nguồn dữ liệu | 2 |
| [3](#page5) | | | Fox’s Algorithm for matrix multiplication | 2 |
| 4 | | | Speed up |  |
| 5 | | | MPI one-sided communication |  |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  |
|  | |  | |  |
|  | |  | |  |
|  | |  | |  |
|  | |  | |  |
|  | |  | |  |
|  | |  | |  |
|  | |  | |  |
|  | |  | |  |
|  | |  | |  |

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

1 . Giới thiệu

The Message Passing Interface Standard (MPI) là một tiêu chuẩn thông qua thư viện dựa trên sự đồng thuận của Diễn đàn MPI, có hơn 40 tổ chức tham gia, bao gồm các nhà cung cấp, nhà nghiên cứu, nhà phát triển thư viện phần mềm và người dùng. Mục tiêu của MPI là thiết lập một tiêu chuẩn di động, hiệu quả và linh hoạt để giao tiếp trong các mô hình song song. Những ưu điểm của MPI:

* Tiêu chuẩn hóa - MPI là thư viện thông qua duy nhất có thể được coi là một tiêu chuẩn. Nó được hỗ trợ trên hầu hết tất cả các nền tảng HPC.
* Tính di động - Có rất ít hoặc không cần phải sửa đổi mã nguồn của bạn khi bạn chuyển ứng dụng của mình sang một nền tảng khác hỗ trợ (và tuân thủ) tiêu chuẩn MPI.
* Hiệu suất - Việc triển khai MPI là của từng nhà cung cấp sẽ giúp khai thác các tính năng phần cứng gốc để tối ưu hóa hiệu suất. Việc hiện thực MPI là miễn phí.
* Chức năng - Có hơn 430 dịch vụ được xác định trong MPI-3, bao gồm phần lớn các thói quen trong MPI-2 và MPI-1.

Chúng ta hãy tìm hiểu về sử dụng MPI để xử lý bài toán về ma trận kích thước lớn.

Đề tài 7: Viết chương trình nhân ma trận kích thước 1.000x1.000, 10.000x10.000 và 100.000x100.000 (có trao đổi hàng/cột giữa các bộ xử lý) trên hệ thống máy tính ảo có giao tiếp 1Gpbs, 10 Gbps (Gigabit Ethernet), 40 Gbps (Infiniband).

Assignment - Parallel Processing Course, 2019-2020 Page 1

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

2 . Nguồn dữ liệu

Dữ liệu là 3 cặp ma trận lần lượt có kích thước 1.000x1.000, 10.000x10.000, 100.000x100.000 được tạo bằng C++:

#define N 100000 //N lan luot la 1000, 10000, 100000

int main( int argc, char\* argv[] ){

ofstream myfile;

myfile.open ("100000.txt");

for (int i = 0; i<N; i++){

for(int j = 0; j < N-1; j++)

myfile << "1 ";

myfile << "1\n";

}

myfile.close();

return 0;

}

Để thuận tiện cho việc quan sát kết quả, các ma trận được tạo ra được lưu

trong file .txt và có tất cả các phần tử đều là 1.

Kết quả tạo ma trận được lưu tại link sau:

<https://tinyurl.com/matrix27>

3 . Fox’s Algorithm for matrix multiplication

In order to simplify the discussion, let’s assume( for the time being ) that the matrixces have order n, and the number of processes, p, equals . Then a checkerboard mapping assigns , to process i\*n + j, or, loosely, process (I,j). Fox’s algorithm for matrix multiplication proceeds in n stages. One stage for each term in dot product

During the initial stage, each process multiplies the diagonal entry of A in its process row by its element of B:

Stage 0 on process(I,j) : **.**

During the next stage, each process multiplies the element immediately to the right of the diagonal of A (in its process row) by the element of B directly beneath its own element of B:

Stage 1 on process (I,j): **.**

In general, during the kth stage, each process multiplies the element k columns to the right of the diagonal of A by the element k rows below its own element of B:

Stage k on process (I,j):  **.**

Assignment - Parallel Processing Course, 2019-2020 Page 2

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

Of course, we can’t just add k to a row or column subscript and expect to always get a valid row or column number. One possible solution is to use subscripts modulo n. That is, rather than use i+k for a row or column subscript, use i+k mod n. Then, we will get a valid pair of subscripts:

Stage k on process(I,j) :  **= (i+k) mod n; .**

Also observe that we’ll compute :

***= + +***

In general, during the kth stage, each process in the ith row uses . Where = (i+k) mod n. Thus, we need to broadcast across the ith row before each multiplication. Finally observe that during the initial stage, each process uses its own element of B. During subsequent stages, process (I,j) will use . Thus, after each multiplication is completed, the elements of B should be “shifted” up one row, and elements in the top row should be sent to the bottom row.

However, that we’re unlikely to have access to processors even for relatively small matrices. So how can we modify our algorithm so that it use fewer than processors.

* Carry out the algorithm we have just outlined using submatrices. One way we can insure that this is the case is to use a square grid of processes, where the number of process rows or process columns, , evenly divides n. With this assumption, each process is assigned a square n/ submatrix of each of the three matrices. Specifically, let : = n/ and define to be the submatrix of A.
* We make similar definitions of . Assign all to process (I,j). and define q = , Then our algorithm will compute:

**= + +**

Assignment - Parallel Processing Course, 2019-2020 Page 3

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

* Implementation of Fox’s Algorithm

First, we’ll write a functin that creates the various communicators and associated information.

|  |  |
| --- | --- |
| struct grid\_info { | |
| int nr\_world\_processes; |
| MPI\_Comm comm; |
| MPI\_Comm row\_comm; |
| MPI\_Comm col\_comm; |
| int ppside; |
| int my\_row, my\_col, my\_rank, my\_world\_rank; |
| }; | |
| |  | | --- | |  | | void grid\_setup(struct grid\_info \*grid) { | |  | | | | MPI\_Comm\_size(MPI\_COMM\_WORLD, &(grid->nr\_world\_processes)); | | | | MPI\_Comm\_rank(MPI\_COMM\_WORLD, &(grid->my\_world\_rank)); | | | | grid->ppside = intsqrt(grid->nr\_world\_processes); | | | | int dimensions[2] = {grid->ppside, grid->ppside}; | | | int wrap\_around[2] = {TRUE, TRUE}; | | | int reorder = TRUE; | | | MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dimensions, wrap\_around, reorder, &(grid->comm)); | | | MPI\_Comm\_rank(grid->comm, &(grid->my\_rank)); | |  | | int coordinates[2]; | | MPI\_Cart\_coords(grid->comm, grid->my\_rank, 2, coordinates); | | grid->my\_row = coordinates[0]; | | grid->my\_col = coordinates[1]; | |  | | int free\_coords\_for\_rows[] = {FALSE, TRUE}; | | int free\_coords\_for\_cols[] = {TRUE, FALSE}; | | MPI\_Cart\_sub(grid->comm, free\_coords\_for\_rows, &(grid->row\_comm)); | | MPI\_Cart\_sub(grid->comm, free\_coords\_for\_cols, &(grid->col\_comm)); | | } | |  |  | | | | |  |

Assignment - Parallel Processing Course, 2019-2020 Page 4

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

Now let’s write the function that does the actual multiplication. We’ll assume that the user has supplied the type definitions and functions for the local matrices. Specifically, we’ll assume user has supplied a type definition for LOCAL\_MATRIX\_T and corresponding derived type, Local\_matrix\_mpi\_t and three functions: Local\_matrix\_multiply, Local\_matrix\_allocate and Set\_to\_zero. We also assume that storage for the parameters has been allocated in the calling function, and all the parameters, except the product matrix local\_C, have been initialized.

|  |
| --- |
| void Fox(struct grid\_info \*grid, int local\_n, |
| matrix\_type\*\* local\_A, matrix\_type\*\* local\_B, matrix\_type\*\* local\_C) |
| { |
| int stage; |
| int i, j; |
| const int local\_n\_sq = local\_n \* local\_n; |
| const int src = (grid->my\_row + 1) % grid->ppside; |
| const int dest = (grid->my\_row + grid->ppside - 1) % grid->ppside; |
| int bcast\_root; |
| MPI\_Status status; |
| matrix\_type \*\*temp\_A = matrix\_new(local\_n, local\_n); |
|  |
| for (stage = 0; stage < grid->ppside; ++stage) { |
| bcast\_root = (grid->my\_row + stage) % grid->ppside; |
| if (bcast\_root == grid->my\_col) { |
| MPI\_Bcast(\*local\_A, local\_n \* local\_n, MPI\_FLOAT, bcast\_root, grid->row\_comm); |
| matrix\_multiply\_and\_add(local\_A, local\_B, local\_C, local\_n, local\_n, local\_n); |
| } |
| else { |
| MPI\_Bcast(\*temp\_A, local\_n\_sq, MPI\_FLOAT, bcast\_root, grid->row\_comm); |
| matrix\_multiply\_and\_add(temp\_A, local\_B, local\_C, local\_n, local\_n, local\_n); |
| } |
| MPI\_Sendrecv\_replace(\*local\_B, local\_n\_sq, MPI\_FLOAT, dest, 0, src, 0, grid->col\_comm, &status); |
| } |
| } |

Assignment - Parallel Processing Course, 2019-2020 Page 5

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

4 . Speed up

* **Speed-up for Physical Computer using different number of processors**

**Table 1.** The Results of the computational experiments for estimating the Fox parallel algorithm efficiency using physical computers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Matrix Size | Serial Algorithm (s) | page13image2156213424  Parallel Algorithm | | | |
| 4 processors | | 9 processors | |
| Time (s) | Speed Up | Time | Speed Up |
| 500 | 0,8527 | 0,2190 | 3.8925 | 0,1468 | 5.8079 |
| **1000** | **12,8787** | **3,0910** | **4.1664** | **2,1565** | **5.9719** |
| 1500 | 43,4731 | 10,8678 | 4.0001 | 7,2502 | 5.9960 |
| 2000 | 103,0561 | 24,1421 | 4.2687 | 21,4157 | 4.8121 |
| 2500 | 201,2915 | 51,4735 | 3.9105 | 41,2159 | 4.8838 |
| 3000 | 347,8434 | 87,0538 | 3.9957 | 58,2022 | 5.9764 |
| **10000** | **11371,3702** | **3002,5798** | **3.7872** | **1732,7582** | **6.5626** |
| **100000** | **120103192** | **30025798(estimate)** | **4.00** | **20017198(estimate)** | **6.00** |

Assignment - Parallel Processing Course, 2019-2020 Page 6

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

* **Speed-up for Cluster.**

**Table 2.** The Results of the computational experiments for estimating the Fox parallel algorithm efficiency using (HCMUT-HPC Lab) Linux cluster.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Matrix Size | Serial Algorithm (s) | page13image2156213424  Parallel Algorithm | | | |
| 4 nodes | | 8 nodes | |
| Time (s) | Speed Up | Time | Speed Up |
| 500 | 0.601031 | 0.28615563 | 2.10036406 | 0.277821 | 2.1634 |
| **1000** | **6.69186** | **3.291** | 2.03338195 | **1.8377** | **3.6414** |
| 1500 | 12.0819 | 7.33271118 | 1.64767161 | 3.19091 | 3.7863 |
| 2000 | 72.8571 | 41.5928808 | 1.75167237 | 18.0996 | 4.0253 |
| 2500 | 74.7383 | 43.2956988 | 1.72622921 | 18.8406 | 3.9669 |
| **10000** | **13008.64118** | **5999.98** | 2.16811409 | **2998.5768** | **4.338271803** |
| **100000** | **1200775.238 (estimate)** | **600387.619 (estimate)** | **2.00** | **300427.93 (estimate)** | **4.00** |

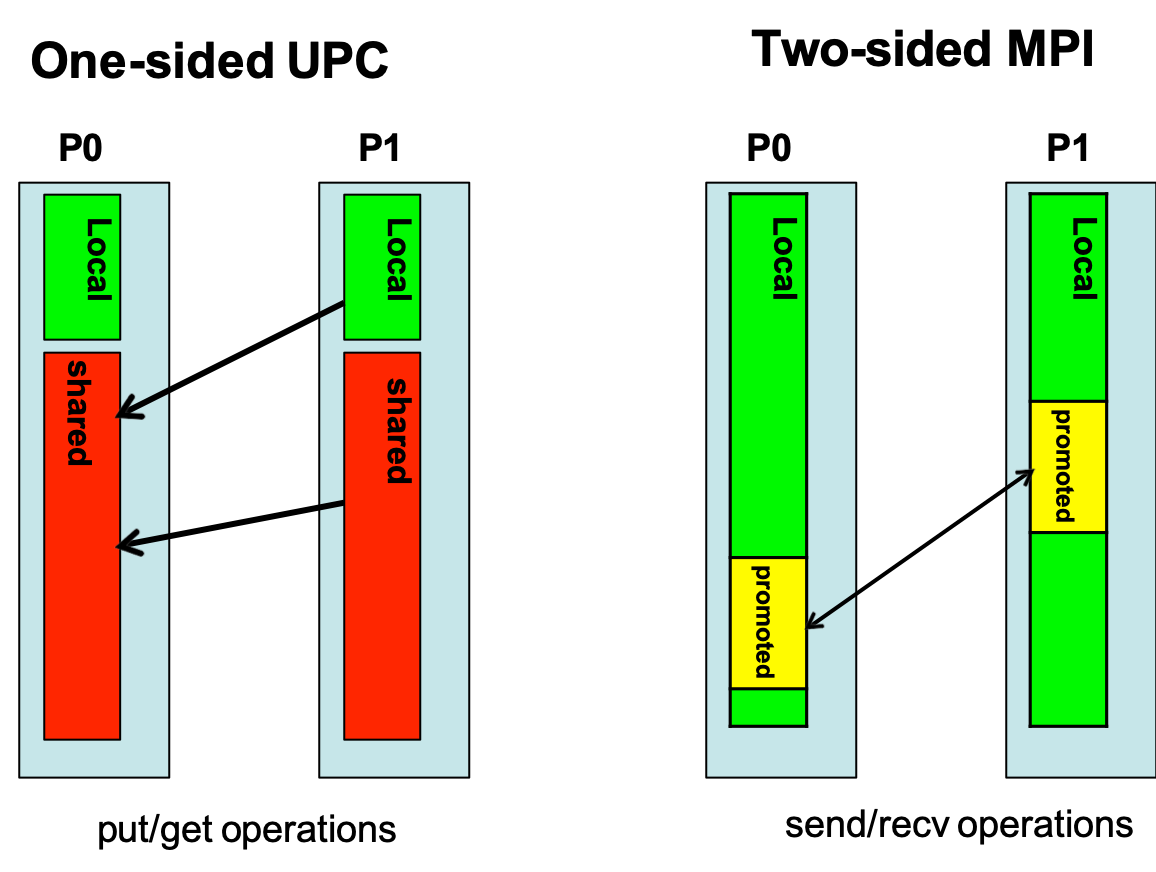
Assignment - Parallel Processing Course, 2019-2020 Page 7

Trường Đại Học Bách Khoa ĐHQG TPHCM

Khoa Khoa Học và Kỹ Thuật Máy Tính

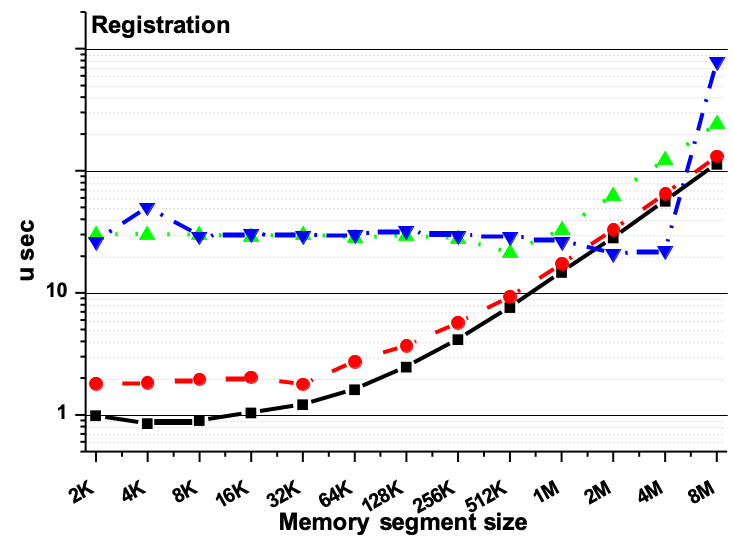
5 . One-Sided Communicatin

1. **Two-Sided Communication vs One-Sided Communication**
   1. Communication Paradigm Models:



* 1. **Registration (Resolving remote ambiguity)**

Assignment - Parallel Processing Course, 2019-2020 Page 8



* Prevents memory swapping
* Allows offloading communication to NIC
* Allows out-of-order progress of many transferExpensive (hopefully not frequent)
* Limited and shared resources
  1. **Two-sided Summary of Challenges**
* Remote involvement
  + Rank-to-rank strict ordering can cause
    - Node-to-node strict ordering by hardware
    - How to handle message cancellation?!
  + Message size ambiguity
    - int MPI\_Irecv(buf, **count**,datatype, source, tag, …)
      * count reflects buffer size **NOT** msg size.
      * Receiver cannot tell what protocol will be used before matching
    - Probing obstruct relaxed ordering progress.
* Remote readiness for transfer
  + All memory space default to local
  + Registration is on-demand
    - Variability of performance based on hit/miss in registration cache.
  1. **Differences and Similarities between one-sided and two-sided communication:**

Assignment - Parallel Processing Course, 2019-2020 Page 9

|  |  |
| --- | --- |
| **Differences** | **Similarities** |
| Language semantic | Runtime optimization |
| Ability to exploit relaxed ordering | Registration overhead or, Copying |

* 1. **Conclusion**:
* One-sided communication exploits relaxed ordering with ease
  + Communicate-able memory is easier to identify
    - annotated shared - can be registered upfront.
  + Relaxed vs. strict ordering is explicitly specified
    - by programmer (or programming language). (Default to relaxed)
  + Large percentage of the peak performance for different communication patterns
* Two-sided faces the following challenges
  + Strict matching between send and receives (large startup overhead)
  + Locality notion (everything default to local)
    - Expensive to prepare buffer for communication (registration).
  + Receiver ambiguity about transactions (probing or over allocation)
  + Performance may need high node concurrency (problematic to in-node communication).

**REFERENCES:**

1. LAWRENCE BERKELEY NATIONAL LABORATORY - <https://www.lbl.gov>
2. WIKIPEDIA – <https://wikipedia.org>

3. <https://archive.org/details/parallelprogramm00pach/page/126>

Assignment - Parallel Processing Course, 2019-2020 Page 10