Параллельное программирование для высокопроизводительных систем

кафедра СКИ
сентябрь – декабрь 2021 г.

Лектор доцент Н.Н.Попова

Лекция 11
18 ноября 2021 г.

Тема

- Параллельные алгоритмы матричного умножения. Алгоритмы Кеннона и Фокса.
- Модель Хокни оценки коммуникационных операций.
- Эффективность алгоритма Кеннона.

Идея алгоритма Кеннона

- Алгоритм Кеннона определяет порядок суммирования членов во внутреннем цикле:

$$C(i,j) = \sum_{k=0}^{s-1} A(i, (i + j + k) \bmod s) * B((i + j + k) \bmod s, j)$$

таким образом, чтобы в каждом процессе на каждом шаге алгоритма находился один из блоков матриц A и B. Предусматривается первоначальное распределение блоков матриц таким образом, чтобы минимизировать обмены блоками в процессе выполнения алгоритма.

Алгоритм Кеннона

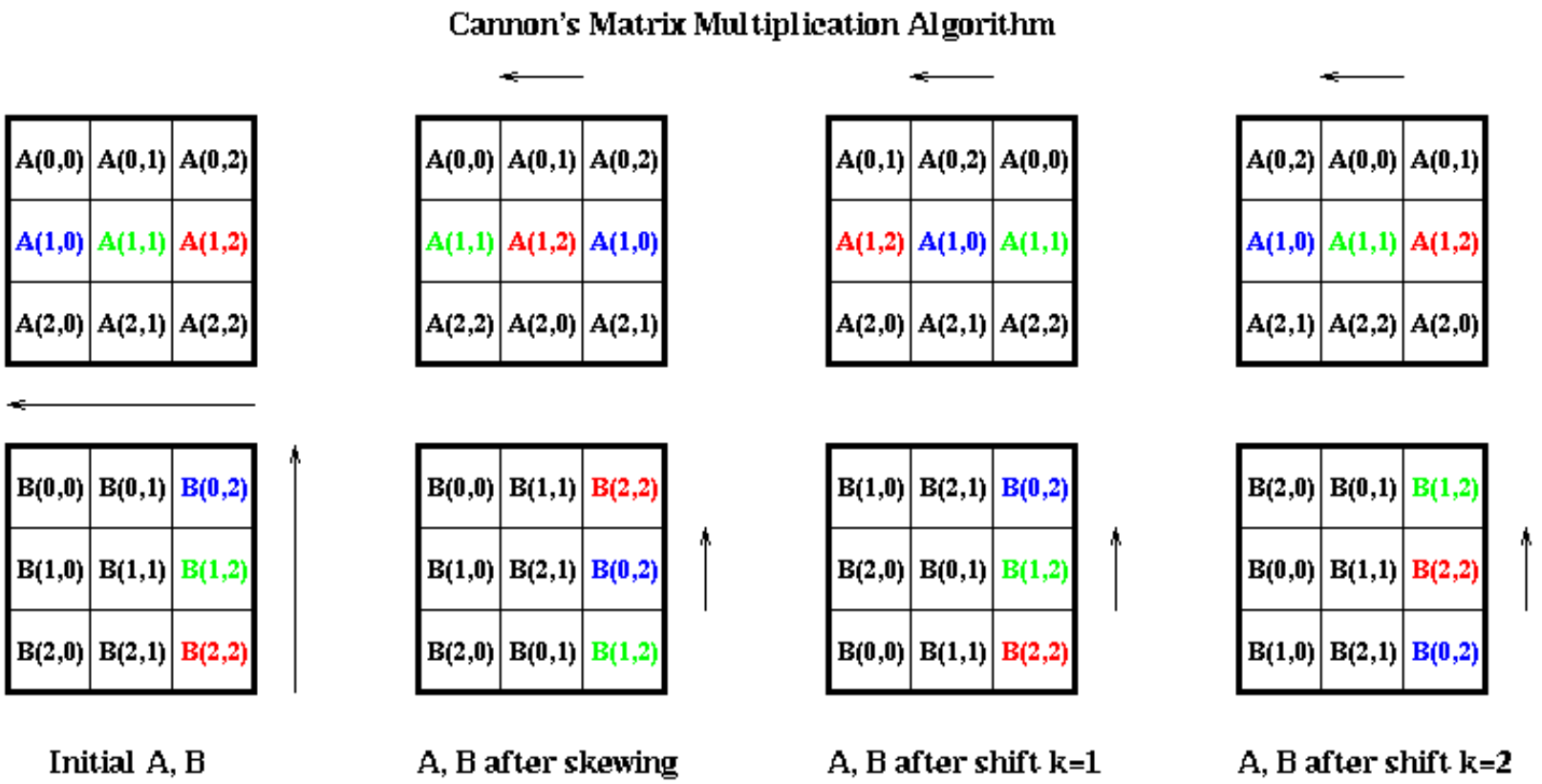


Схема алгоритма Кеннона

for all (i=0 to s-1) // начальное распределение блоков матрицы A

Циклический сдвиг влево строки i матрицы A на j

так, чтобы на место A(i,j) была записана подматрица A(i,(i+j) mod s

end

for all (i=0 to s-1) // начальное распределение блоков матрицы B

Циклический сдвиг вверх столбца j матрицы B на j

так, чтобы на место B(i,j) была записана подматрица B((i+j) mod s,j)

end

for k=0 to s-1

for all (i=0 to s-1, j=0 to s-1)

C(i,j) = C(i,j) + A(i,j)*B(i,j)

Схема алгоритма Кеннона

Циклический сдвиг влево каждой строки матрицы A на 1 так, чтобы на место A(i,j) была записана подматрица A(i,(j+1) mod s)

Циклический сдвиг вверх каждого столбца матрицы B на 1 так, чтобы на место B(i,j) была записана подматрица B((i+1) mod s,j)

end for

end for

Алгоритм Кеннона: основной цикл

```
// определение процессной решетки
dims[0] = dims[1] = sqrt(P);
periods[0] = periods[1] = 1;
MPI_Cart_Create(comm,2,dims,periods,1,&comm_2d);

// определение координат процесса в решетке
MPI_Comm_rank(comm_2d, &my2drank);
MPI_Cart_coords(comm_2d, my2drank, 2, mycoords);

// определение соседей в решетке
MPI_Cart_shift(comm_2d, 0, -1, &rightrank, &leftrank);
MPI_Cart_shift(comm_2d, 1, -1, &downrank, &uprank);

nlocal = n/dims[0];
```

Алгоритм Кеннона: основной цикл

```
/* Начальное распределение блоков матрицы A:
   сдвиг влево на i позиций */
MPI_Cart_shift(comm_2d, 0, -mycoords[0], &shiftsource, &shiftdest);
MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE, shiftdest, 1,
      shiftsource, 1, comm_2d, &status);

/* Начальное распределение блоков матрицы B:
   сдвиг вверх на j позиций */
MPI_Cart_shift(comm_2d, 1, -mycoords[1], &shiftsource, &shiftdest);
MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE, shiftdest, 1,
      shiftsource, 1, comm_2d, &status);
```

Алгоритм Кеннона: основной цикл

```
/* Main Computation Loop */
for(i=0; i<dims[0]; i++){
  MatrixMultiply(nlocal,a,b,c); /* c=c+a*b*/

  /* Shift matrix a left by one */
  MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE, leftrank, 1, rightrank,
      1, comm_2d, &status);

  /* Shift matrix b up by one */
  MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE, uprank, 1,
      downrank, 1, comm_2d, &status);
}
```

Алгоритм Кеннона: основной цикл

```
/* Restore original distribution of a and b */
MPI_Cart_shift(comm_2d, 0, +mycoords[0], &shiftsource, &shiftdest);
MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE, shiftdest, 1,
      shiftsource, 1, comm_2d, &status);

MPI_Cart_shift(comm_2d, 1, +mycoords[1], &shiftsource, &shiftdest);
MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE, shiftdest, 1,
      shiftsource, 1, comm_2d, &status);
```

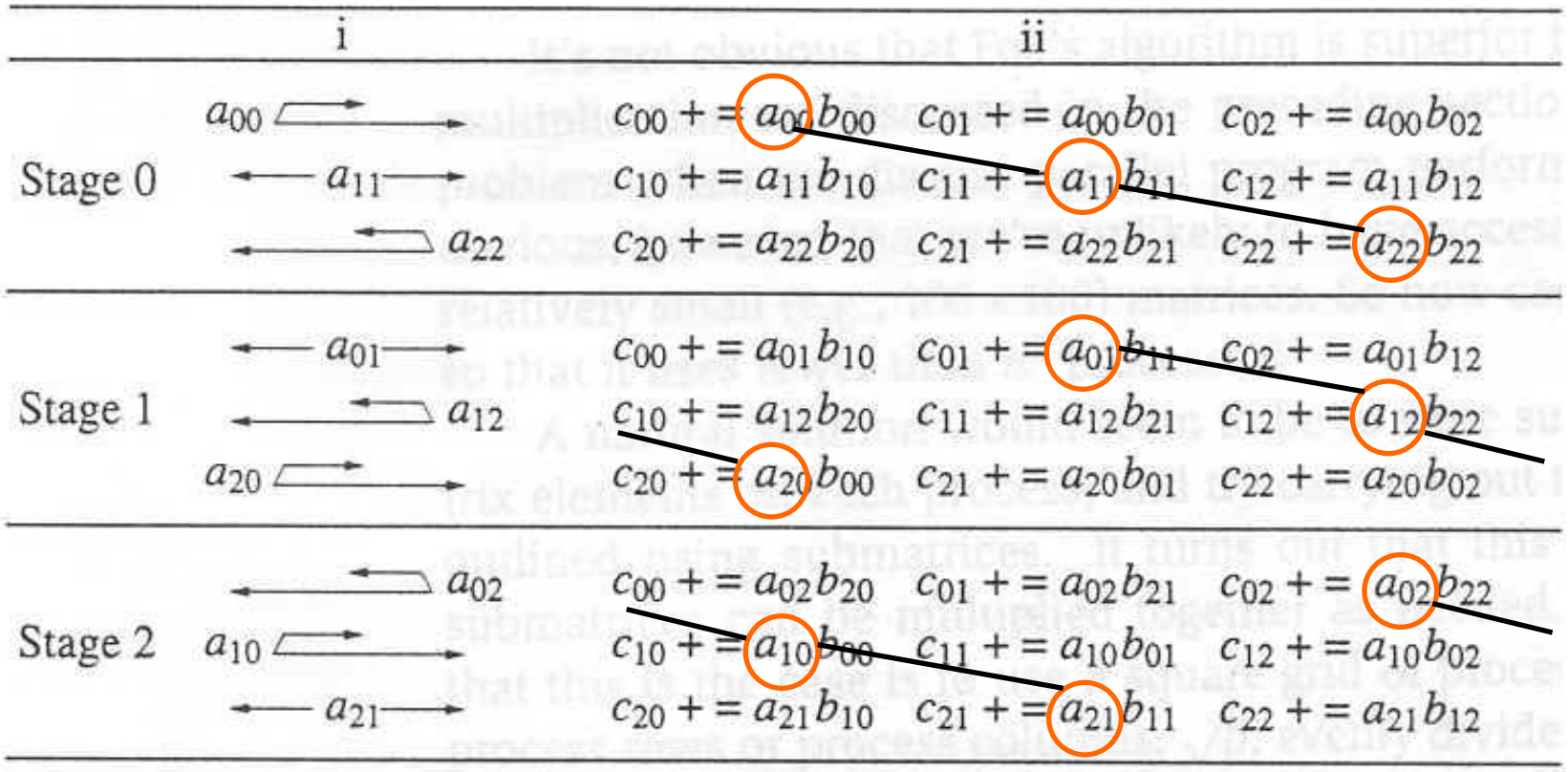
- Алгоритм Фокса

Pacheco,Peter Parallel programing with MPI.

Блочный алгоритм Фокса

- Аналогично алгоритму Кеннона: создание 2-мерной процессной решетки, распределение блоков матриц A, B, C по процессам
- Основная идея: вместо начального перераспределения всех блоков матриц по процессам используется диагональный блок матрицы A процессной строки, в которой находится этот элемент, и находящийся в процессе блок матрицы B.
- Далее на каждом шаге алгоритма по процессной строке широковещательной передачей раздается очередной блок матрицы A и блок матрицы B, как в алгоритме Кеннона
- При оценке времени выполнения время передачи между соседними процессами блока матрицы A, заменяется на широковещательную операцию рассылки блока по $\sqrt{P}$ процессам

Алгоритм Фокса



Алгоритм Фокса

$$C_{ij} = A_{io}B_{oj} + A_{i1}B_{1j} + A_{i2}B_{2j} \ldots A_{i,n-1}B_{n-1,\,j}$$

- Шаг 0

  Процесс(i, j) : $C_{ij} = A_{ii} \times B_{ij}$

- Шаг 1

  Процесс(i, j) : $C_{ij} = C_{ij} + A_{i,i+1} \times B_{i+1,\,j}$

- Шаг k

  Процесс(i, j) : $C_{ij} = C_{ij} + A_{i,i+k} \times B_{i+k,\,j}$

Алгоритм Фокса

Шаг 1. **Широковещательная рассылка диагонального** элемента каждой строки матрицы A по всем процессорам своей строки.

Каждый процессор (i,j) выполняет

C(i,j) = A(i,i)*B(i,j)

Столбец матрицы B циклически сдвигается вверх по своему столбцу, замещая элемент B(i,j).

Шаг 2. **Широковещательная рассылка элемента матрицы A, находящегося справа** от диагонального, по всем процессорам своей строки.

Каждый процессор (i,j) выполняет

C(i,j) = C(i,i)+A(i+1,i)*B(i+1,j)

Столбец матрицы **B циклически сдвигается вверх** по своему столбцу

Шаг k. **Широковещательная рассылка очередного (i+k) mod s** элемента строки матрицы A по всем процессорам своей строки.

Каждый процессор (i,j) выполняет:

C(i,j) = C(i,j) + A(i,(i+k) mod s)*B((i+k) mod s,j)

Столбец матрицы **B циклически сдвигается вверх** по своему столбцу, замещая собой текущий элемент B(i,j)

Алгоритм Фокса (1)

```
typedef struct {
  int p; /* Общее число процессов */
  MPI_Comm comm; /* Коммуникатор для сетки */
  MPI_Comm row_comm; /* Коммуникатор строки */
  MPI_Comm col_comm; /* Коммуникатор столбца */
  int q; /* Порядок сетки */
  int my_row; /* Номер строки */
  int my_col; /* Номер столбца */
  int my_rank; /* Ранг процесса в коммуникаторе сетки */
} GRID-INFO-TYPE;
```

Алгоритм Фокса (2)

```
void Setvup_grid(GRID_INFO_TYPE* grid) {
  int old_rank;
  int dimensions[2];
  int periods[2];
  int coordinates[2];
  int varying-coords[2];
  /* Настройка глобальной информации о сетке */
  MPI_Comm_size(MPI_COMM_WORLD, &(grid->p));
  MPI_Comm_rank(MPI_COMM_WORLD, &old_rank);
  grid->q = (int) sqrt((double) grid->p);
  dimensions[0] = dimensions[1] = grid->q;
  periods[0] = periods[1] = 1;
  MPI_Cart_create(MPI_COMM_WORLD, 2, dimensions,
     periods, 1,    &(grid->comm));
```

Алгоритм Фокса (3)

```
MPI_Comm_rank(grid->comm, &(grid->my_rank));
MPI_Cart_coords(grid->comm, grid->my_rank, 2, coordinates);
grid->my_row = coordinates[0];
grid->my_col = coordinates[1];
 /* Настройка коммуникаторов для строк и столбцов */
  varying_coords[0] = 0; varying_coords[1] = 1;
  MPI_Cart_sub(grid->comm, varying_coords, &(grid-
     >row_comm));
  varying_coords[0] = 1; varying_coords[1] = 0;
  MPI_Cart_sub(grid->comm, varying_coords, &(grid-
     >col_comm));
} /* Setup_grid */
```

Алгоритм Фокса (4)

```
void Fox(int n, GRID_INFO_TYPE* grid, LOCAL_MATRIX_TYPE*
    local_A, LOCAL_MATRIX_TYPE* local_B,
    LOCAL_MATRIX_TYPE* local_C)
{
  LOCAL_MATRIX_TYPE* temp_A;
  int step;
  int bcast_root;
  int n_bar; /* порядок подматрицы = n/q */
  int source;
  int dest;
  int tag = 43;
  MPI_Status status;
  n_bar = n/grid->q;
  Set_to_zero(local_C);
```

Алгоритм Фокса (5)

```
/* Вычисление адресов для циклического сдвига B */
  source = (grid->my_row + 1) % grid->q;
  dest = (grid->my_row + grid->q-1) % grid->q;

  /* Выделение памяти для рассылки блоков A */
  temp_A = Local_matrix_allocate(n_bar);

/* Основной цикл */
  for (step = 0; step < grid->q; step++) {
    bcast_root = (grid->my_row + step) % grid->q;
    if (bcast_root == grid->my_col) {
      MPI_Bcast(local_A, 1, DERIVED_LOCAL_MATRIX,
      bcast_root,   grid->row_comm);
      Local_matrix_multiply(local_A, local_B, local_C);
```

Алгоритм Фокса (6)

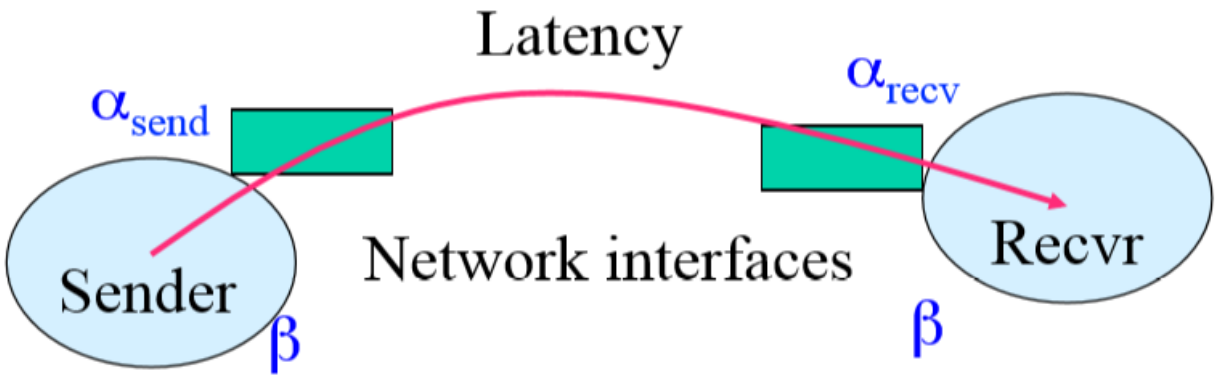
```
} else {
     MPI_Bcast(temp_A, 1, DERIVED_LOCAL_MATRIX,
    bcast_root,
        grid->row_comm);
     Local_matrix_multiply(temp_A, local_B, local_C);
 }
 MPI_Send(local_B, 1, DERIVED_LOCAL_MATRIX, dest, tag,
   grid->col_comm);
 MPI_Recv(local_B, 1, DERIVED_LOCAL_MATRIX, source, tag,
   grid->col_comm, &status);

 } /*for*/

}/*Fox*/
```

Модели оценки коммуникационных операций

- Модель Хокни (1994)

  R. W. Hockney.The communication challenge for mpp : Intel paragon and meiko cs-2.Parallel Computing, 20:389–398, 1994.

- Упрощения: сообщения – непрерывные, в сети нет других передач, сообщение передается непосредственно в буфер в памяти принимающего процесса



Latency & Bandwith

- Модель:

  - в случае коротких сообщений во времени передачи доминирует latency

  - в случае длинных сообщений - bandwith

- Critical message size = latency * bandwith

Производительность двухточечных операций MPI. Модель Хокни.

Время передачи **= latency + размер сообщения / bandwidth**

α β=1/bandwith

- **Latency** – время запуска обмена = время пересылки нулевого сообщения, не зависит от размера сообщения
- **Bandwith** – пропускная способность,число байт в секунду

- **Цена обмена = latency * bandwith** – число байт, которые могли бы быть переданы за время запуска обмена.
- Если **размер сообщения >> цены обмена** – производительность канала обмена близка к пропускной способности канала

Модели оценки коммуникационных операций. Модель LogP.

- Модель: LogP model (Culler et al, 1993)
- 4 параметра:

  - **L** латентность сети

  - **o** накладные системные расходы (message splitting and packing,buffer management, connection, . . .) для сообщения размера w

  - **g** минимальное время между пересылкой двух пакетов сообщения размера w

  - **P** число процессоров

David Culler, Richard Carp, David Pattwrson et al LogP:Towards a Realistic Model of Parallel Computatio ***Communications of the ACM, Vol. 39, No. 11, pp. 78-85***

Оценка времени выполнения коллективных операций. Линейный алгоритм.

- Простой линейный broadcast:
  – Один процесс рассылает данные всем P процессам, по s байт каждому процессу:

$$T(s) = P * (\alpha+\beta s) = \mathcal{O}(P)$$

Оценка времени выполнения коллективных операций. K-нарное дерево.

- Процесс, рассылающий сообщения (процесс root), рассылает сообщение k соседям, которые дальше рассылают его другим k соседям и т.д.
- Время передачи в этом случае:

$$T(s) = \lceil log_k(P) \rceil \cdot (k-1) \cdot (\alpha + \beta \cdot s) = \mathcal{O}(log(P))$$

Эффективность алгоритма Кеннона

forall i=0 to s-1 … s = sqrt(p)

циклический сдвиг строки i матрицы A на i … $t \le s*(\alpha + \beta*n^2/p)$

forall i=0 to s-1

циклический сдвиг столбца i матрицы B на i … $t \le s*(\alpha + \beta*n^2/p)$

for k=0 to s-1

forall i=0 to s-1 and j=0 to s-1

*C(i,j) = C(i,j) + A(i,j)*B(i,j)* … $t = 2*(n/s)^3 = 2*n^3/p^{3/2}$

left-circular-shift each row of A by 1 … $t = \alpha + \beta*n^2/p$

up-circular-shift each column of B by 1 … $t = \alpha + \beta*n^2/p$

- **Общее время** $\text{Total Time} = 2*n^3/p^{3/2} + 4* s*\alpha + 4*\beta*n^2/p$
- **Эффективность** $\text{Parallel Efficiency} = 2*n3 / (p * \text{Total Time})$
  $= 1/(1 + a * 2*(s/n)3 + b * 2*(s/n))$

  Стремится к 1 при n/s = n/sqrt(p) растет
- **Лучше, чем 1D распределение, при котором Efficiency = 1/(1 + O(p/n))**

Ограничения алгоритмов Фокса и Кеннона

- Трудно обобщаются для случаев:
  - p не полный квадрат
  - A и B не квадратные
  - Размерности A, B не делятся нацело на s=sqrt(p)

- Требуется дополнительная память для хранения копий блоков