Московский Государственный Университет им. М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Суперкомпьютеров и Квантовой Информатики

Спецкурс: системы и средства параллельного программирования.

Отчёт № 5.

Анализ 3D-блочного параллельного алгоритма матричного умножения DNS

Работу выполнил

Федоров В. В.

Москва 2020

Постановка задачи и формат данных.

Задача: Реализовать параллельный алгоритм матричного умножения DNS при помощи технологии MPI и выполнить замеры времени выполнения алгоритма. Также использовать инструменты MPI для параллельного ввода/вывода, замерить связанные с вводом/выводом накладные расходы.

Формат командной строки: <файл с матрицей A> <файл с матрицей B> <файл для записи матрицы C> <файл для вывода времени выполнения и времени ввода-вывода>

Формат файла-матрицы: Матрица представляется в виде бинарного файла следующего формата:

Тип	Значение	Описание
Число типа int	N – натуральное число	Число строк матрицы
Число типа int	M – натуральное число	Число столбцов матрицы
Массив чисел типа double	$N \times M$ элементов	Массив элементов матрицы

Элементы матрицы хранятся построчно.

Описание алгоритма.

Математическая постановка: Результатом умножения матриц $A \in \mathbb{R}^{n \times m}$ и $B \in \mathbb{R}^{m \times r}$ является матрица $C \in \mathbb{R}^{n \times r}$, элементы которой вычисляются по формуле:

$$c_{ij} = \sum_{k=0}^{m-1} a_{ik} b_{kj}; i = 0..n-1; j = 0..r-1$$

Матрицы A, B и C можно разделить на блоки. Каждую из матриц разделим на $p \times p$ блоков. Размерность блоков по координате i будем определять следующей формулой:

$$n_i = \frac{n}{p} + (i < n \pmod p)$$

Аналогично определяются размерности по координатам j и k. Тогда умножение матриц A и B можно представить в блочном виде:

$$C_{ij} = \sum_{k=0}^{p-1} A_{ik} B_{kj}; i, j = 0..p-1$$

Параллельный алгоритм DNS основан следующей идее: каждый процесс получает блок матрицы A и блок матрицы B, а затем перемножает их. Для этого строится трехмерная решетка из процессов размерами $p \times p \times p$. Алгоритм можно описать следующим образом:

Процесс {i,k,0} считывает блок A_{ik}	Процесс {k,j,0} считывает блок A_{kj}
Процесс {i,k,0} отсылает свой блок процессу {i,k,k}	Процесс {k,j,0} отсылает свой блок процессу {k,j,k}
Процесс {i,k,k} рассылает свой блок процессам {i,j,k}, j = 0..p-1	Процесс {k,j,k} рассылает свой блок процессам {i,j,k}, i = 0..p-1
Процесс {i,j,k} имеет блок A_{ik}	Процесс {i,j,k} имеет блок B_{kj}
Процесс {i,j,k} вычисляет блок: $C_{ij}^k = A_{ik} B_{kj}$	

Процесс {i,j,0} производит редукцию: $C_{ij}=\sum_{k=0}^{p-1} C_{ij}^{k}$
Процесс {i,j,0} записывает блок C_{ij}

Анализ данных программы: Для оценки времени работы программы использовалась функция MPI_Wtime.

Верификация: Для проверки корректности работы программы она тестировалась на 5 заранее сгенерированных тестах.

Результаты выполнения.

Результаты:

Замеры времени проводились на системе BlueGene/P. Использовались матрицы следующих размеров: 1024×1024; 2048×2048; 4096×4096. Замеры проводились на решетках следующих размеров: $3\times3\times3$; $4\times4\times4$; $5\times5\times5$. В случае решетки 3x3x3;4x4x4;5x5x5 замеры проводились на двух видах мэппинга (размещения процессов в топологии BlueGene/P — трехмерном торе) — стандартном и случайно сгенерированном.

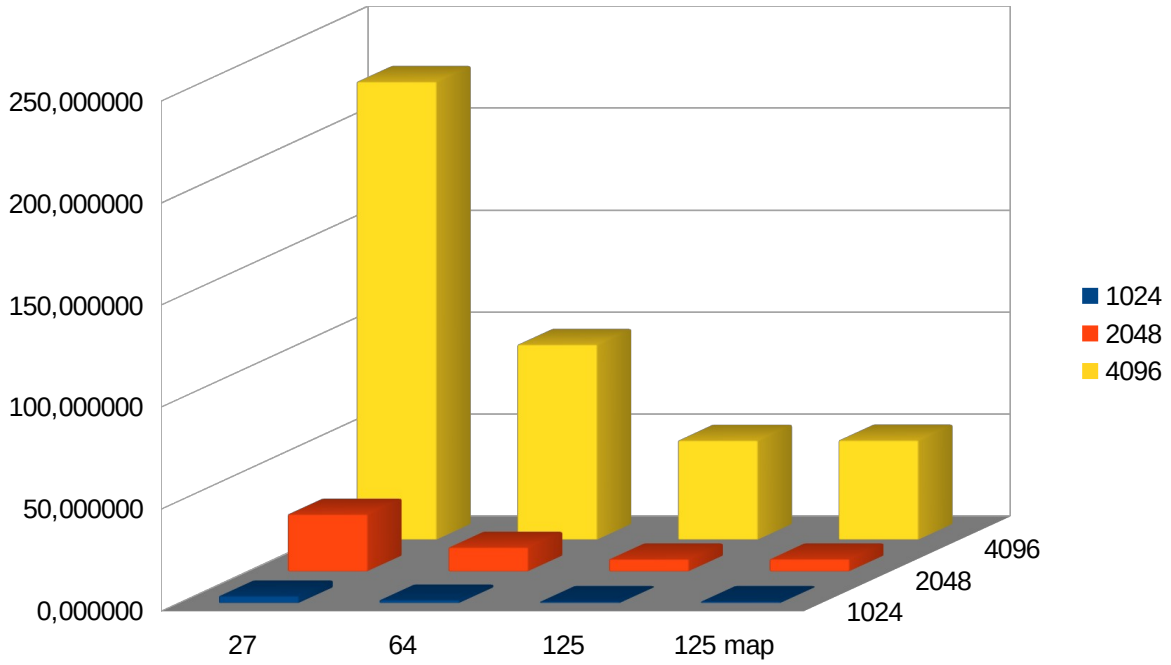
Время выполнения, с				
n	**Число процессов**			
	$3\times3\times3$	$4\times4\times4$	$5\times5\times5$	3x3x3;4x4x4;5x5x5 **map**
1024	3,537664	1,484850	0,768197	0,772243
2048	28,079724	11,875031	6,102440	6,124738
4096	224,575153	95,726594	48,757098	48,838995

Ускорение				
n	**Число процессов**			
	$3\times3\times3$	$4\times4\times4$	$5\times5\times5$	$5\times5\times5$ **map**
1024	1,000000	2,382506	4,605152	4,581024
2048	1,000000	2,364602	4,601393	4,584641
4096	1,000000	2,346006	4,605999	4,598275

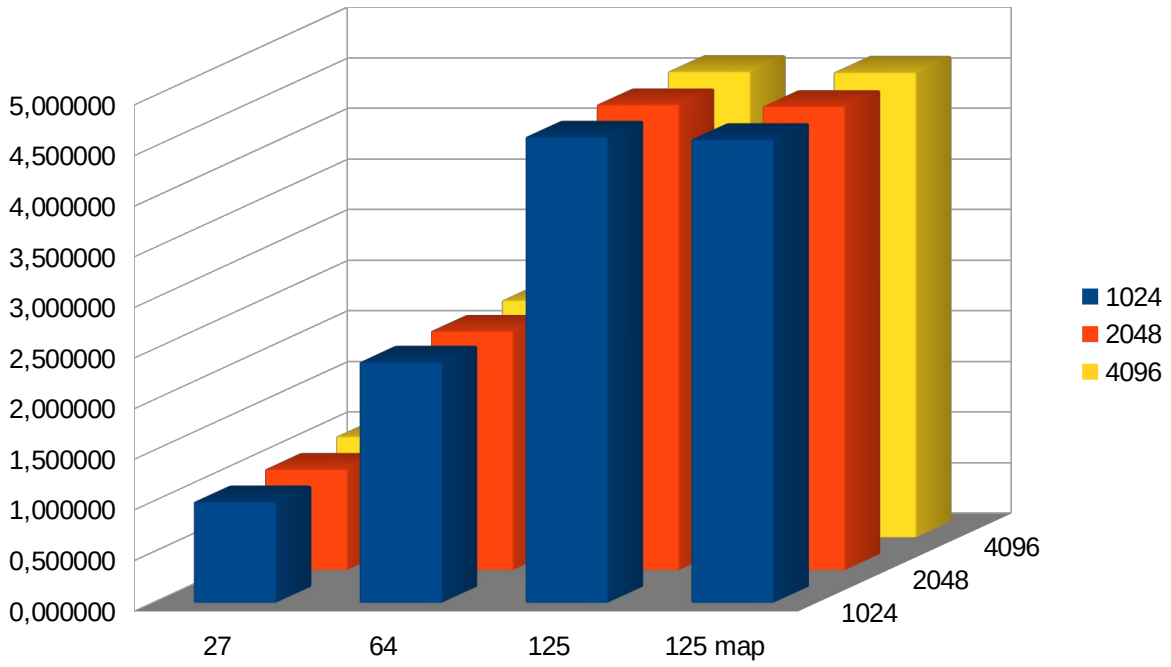
Эффективность				
n	**Число процессов**			
	$3\times3\times3$	$4\times4\times4$	$5\times5\times5$	$5\times5\times5$ **map**
1024	1,000000	1,005120	0,994713	0,989501
2048	1,000000	0,997567	0,993901	0,990282
4096	1,000000	0,989721	0,994896	0,993228

Накладные расходы на ввод-вывод, с				
n	**Число процессов**			
	3×3×3	4×4×4	5×5×5	5×5×5 **map**
1024	0,273663	0,667279	0,430355	0,416688
2048	0,766559	1,486125	0,811998	0,340746
4096	2,391709	1,927498	2,613355	1,492997

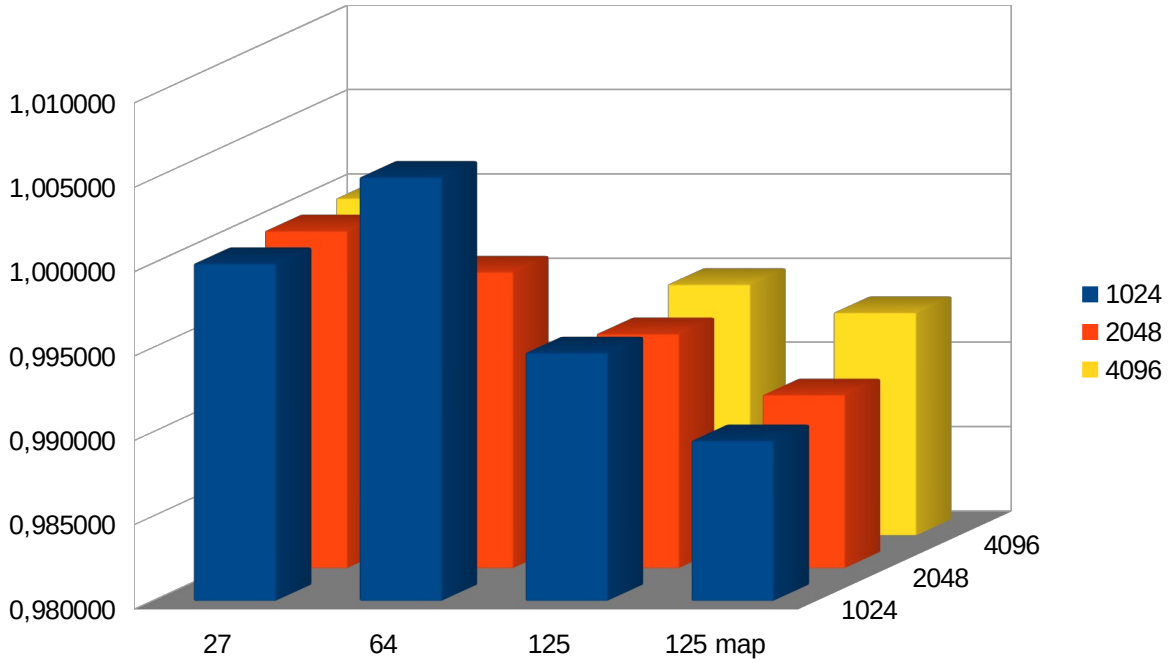
Время выполнения, с



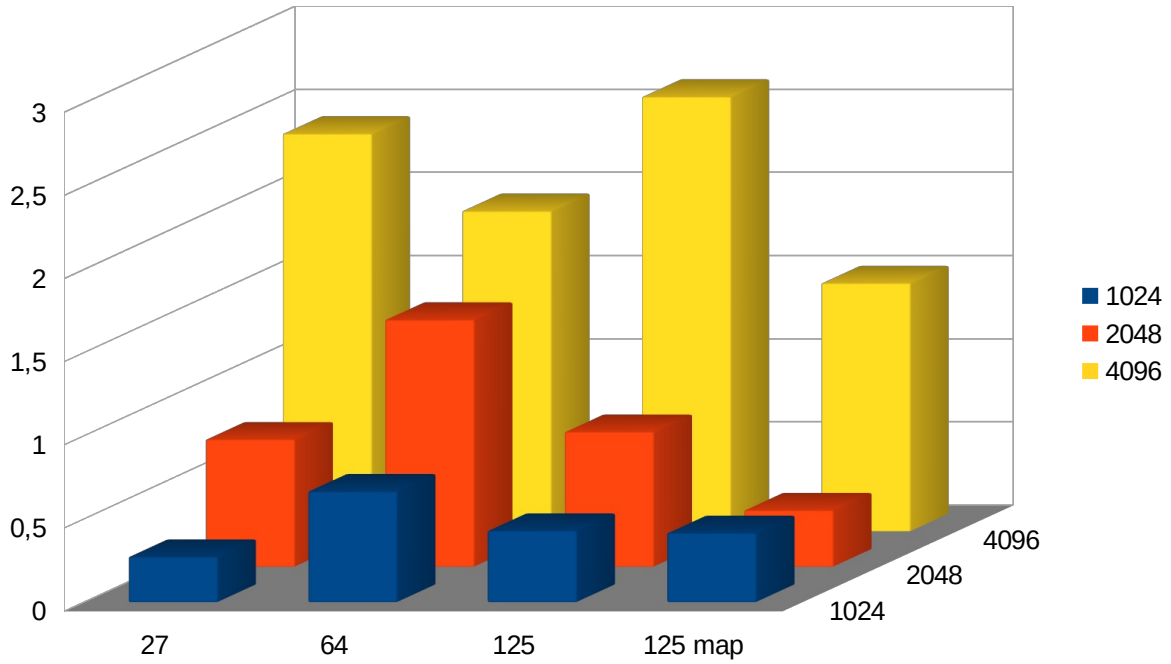
Ускорение



Эффективность



Накладные расходы на ввод-вывод



Основные выводы.

Увеличение числа процессов оказалось максимально продуктивным — эффективность во всех случаях оказалась почти равна 1. Случайный мэппинг не оказал влияния на время работы программы, зато значительно ускорил ввод-вывод.