Разработка параллельной версии программы для задачи Jacobi3D

Report 03.12.17

Гончаренко Дмитрий, 320 группа, МГУ ВМК

Исходные данные: программа jac_3d.c - решающая поставленную задачу Якоби 3D

Задача:

- Реализовать параллельную версию предложенного алгоритма с использованием технологий OpenMP и MPI.
- Исследовать масштабируемость полученной параллельной программы: построить графики зависимости времени исполнения от числа процессоров для различного объема входных данных.
- Для каждого набора входных данных найти количество процессоров, при котором время выполнения задачи перестаёт уменьшаться.

Преобразованная программа:

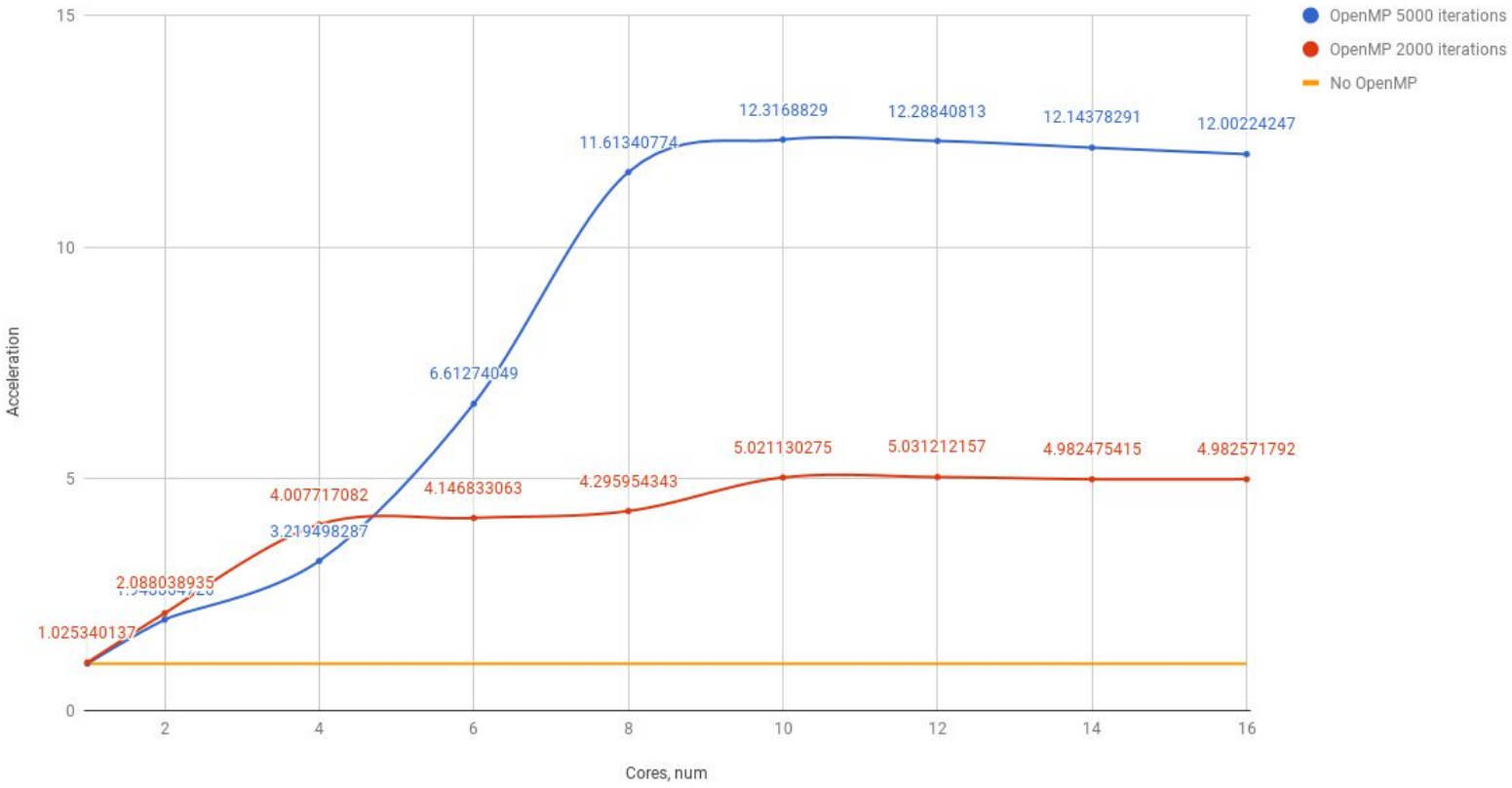
jac3d_omp.c - распараллеленная версия исходной программы с помощью OpenMP.
jac3d_mpi.c - распараллеленная версия с помощью средств MPI.

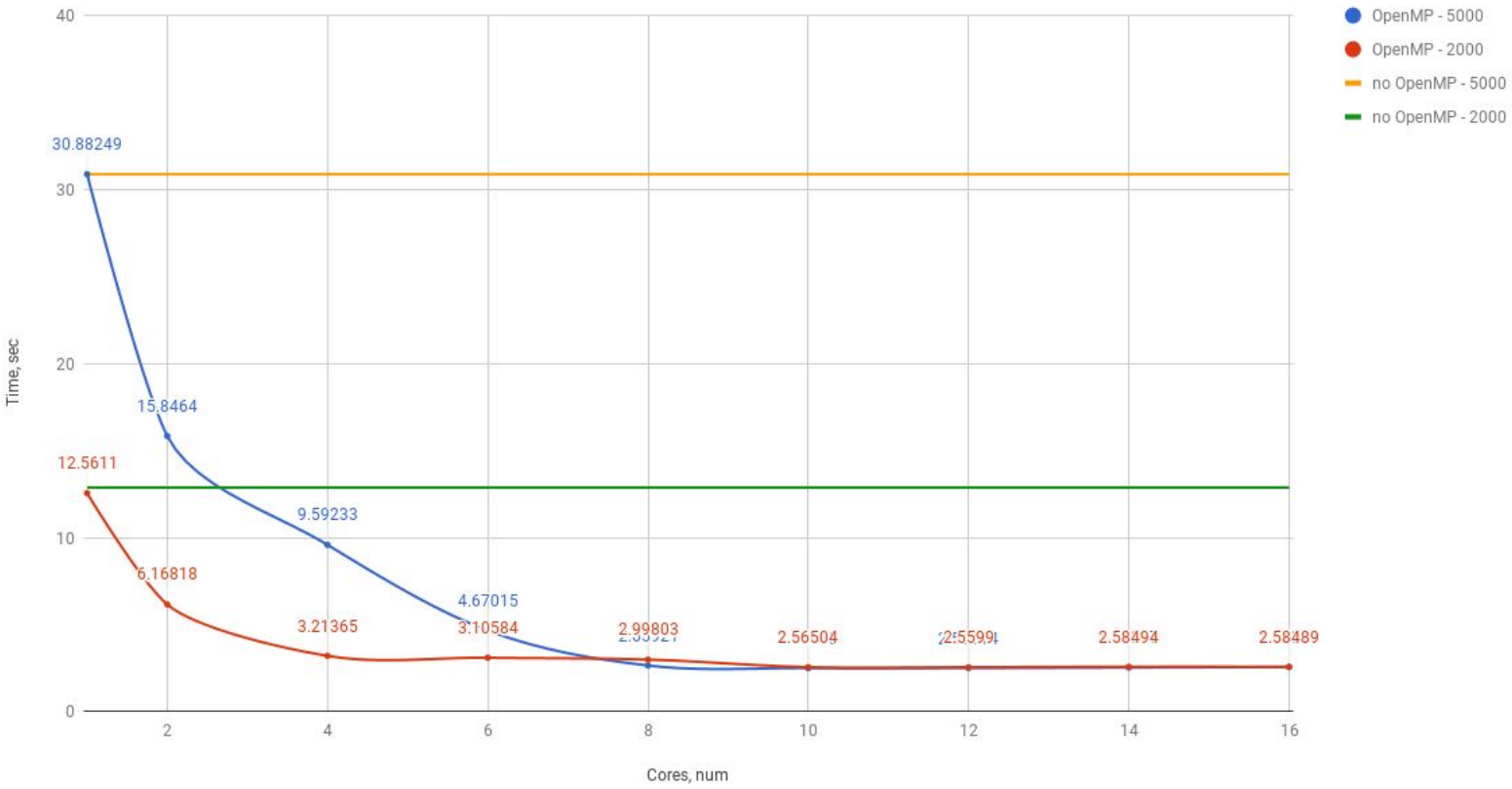
Сводка по OpenMP и MPI

Я воспользовался средством распараллеливания программы на потоки OpenMP:
#pragma parallel for private(i,j)
Конструкция #pragma используется для создания дополнительных указаний компилятору
Ключевая директива omp, указывает что команды будут относится к OpenMP
Директива parallel указывает, что структурный блок, в нашем случае циклы, должен быть выполнен параллельно в несколько потоков. Каждый из потоков выполнит один и тот же участок кода, но разные наборы команд.
Директива for сообщает компилятору, что при выполнении цикла for в блоке итерации должны быть распределены между потоками.
Директива private(i,j) указывает, что данные частные и принадлежат только потоку, то есть переменные i и j будут использоваться потоками по отдельности, без нее переменные начнут использоваться всеми потоками одновременно, что может привести к неверному результату.
Конструкция #pragma critical означает, что разделяемый ресурс, закрывает блок для доступа более чем одного потока одновременно.
#pragma omp for schedule(static) - заранее, до входа в цикл, ставит в соответствие каждому потоку части цикла, которые он будет обрабатывать.
В отличие от OpenMP, где все происходит автоматически на этапе компиляции, в MPI процессам требуется обмениваться информацией вручную. Также OpenMP использует блоки, а MPI работает на протяжении всей программы от начала инициализации до завершения с помощью процессов MPI_Finalize();

Графики OpenMP

После подсчета времени выполнения на ск Regatta, выведем зависимости времени и ускорения от количества используемых в задаче процессоров на основе полученных данных.
Для замера времени использовал вызов функции omp_get_wtime().
На 2000 итерациях не распараллеленного алгоритма Якоби среднее время выполнения в секундах я получил равным 12.8794s. Оптимальный прирост в скорости параллельной программы достигается на 10 процессорах ~ в 5 раз.
Для 5000 итераций: без OpenMP - 30.88249s, оптимальное количество процессоров также 10, прирост более чем в 12 раз. Ниже расположены два графика зависимости ускорения и времени от количества ядер соответственно.

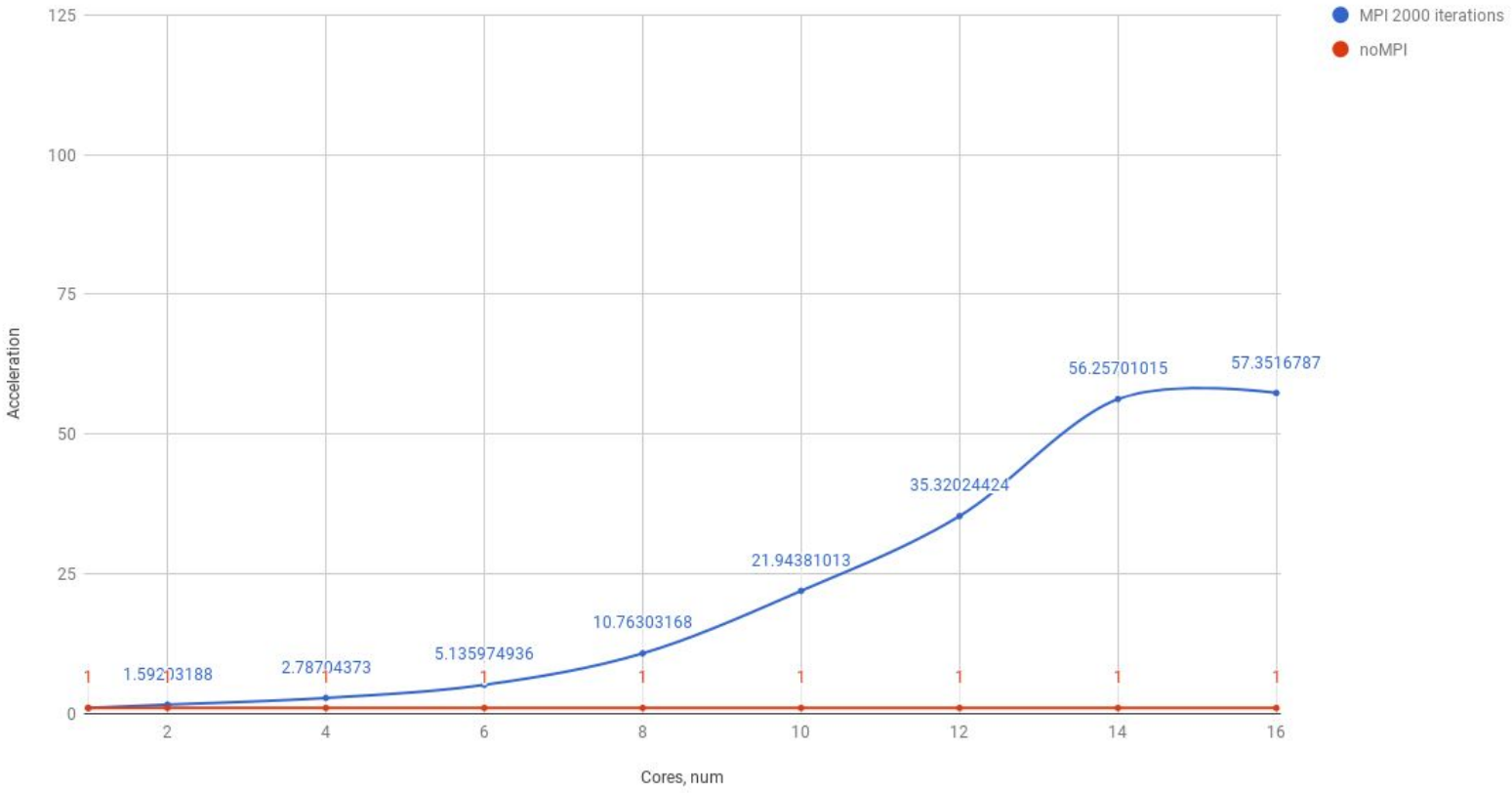


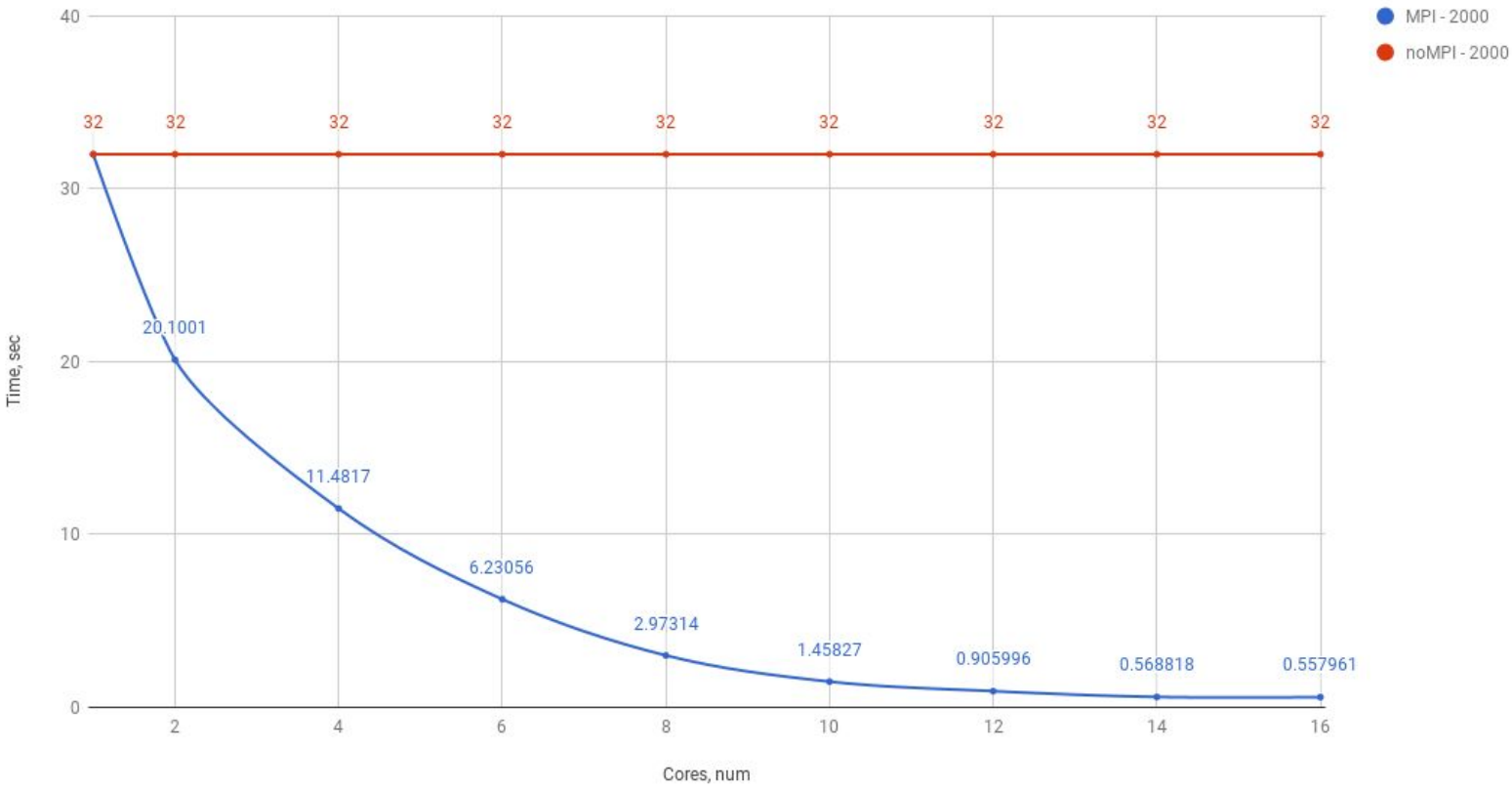


Время выполнения перестает уменьшаться в обоих случаях на 10ти процессорах.

Графики MPI

Подсчет времени выполнения распараллеленной программы с помощью средств MPI я проводил на ск им. М.В. Келдыша РАН. Результаты измерений можно видеть в диаграммах ниже. Замер производился на 2000 итерациях.
По результатам замеров мы получили, что OpenMP существенно ускоряет программу при небольшом количестве процессоров, но его эффективность снижается по мере их увеличения, в отличие от MPI, который дает прирост при любом увеличении процессоров.


MPI 2000 iterations
noMPI
Acceleration
Cores, num
125
100
75
50
25
0
2
4
6
8
10
12
14
16
1
1.59203188
2.78704373
5.135974936
10.76303168
21.94381013
35.32024424
56.25701015
57.3516787


MPI - 2000
noMPI - 2000
Time, sec
Cores, num
40
30
20
10
0
2
4
6
8
10
12
14
16
32
20.1001
11.4817
6.23056
2.97314
1.45827
0.905996
0.568818
0.557961