

Разработка параллельной версии программы для задачи Jacobi3D

Report 03.12.17

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: skipod_jac3d_omp.c - распараллеленная версия исходной программы.

Сводка по OpenMP

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 относиться 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.

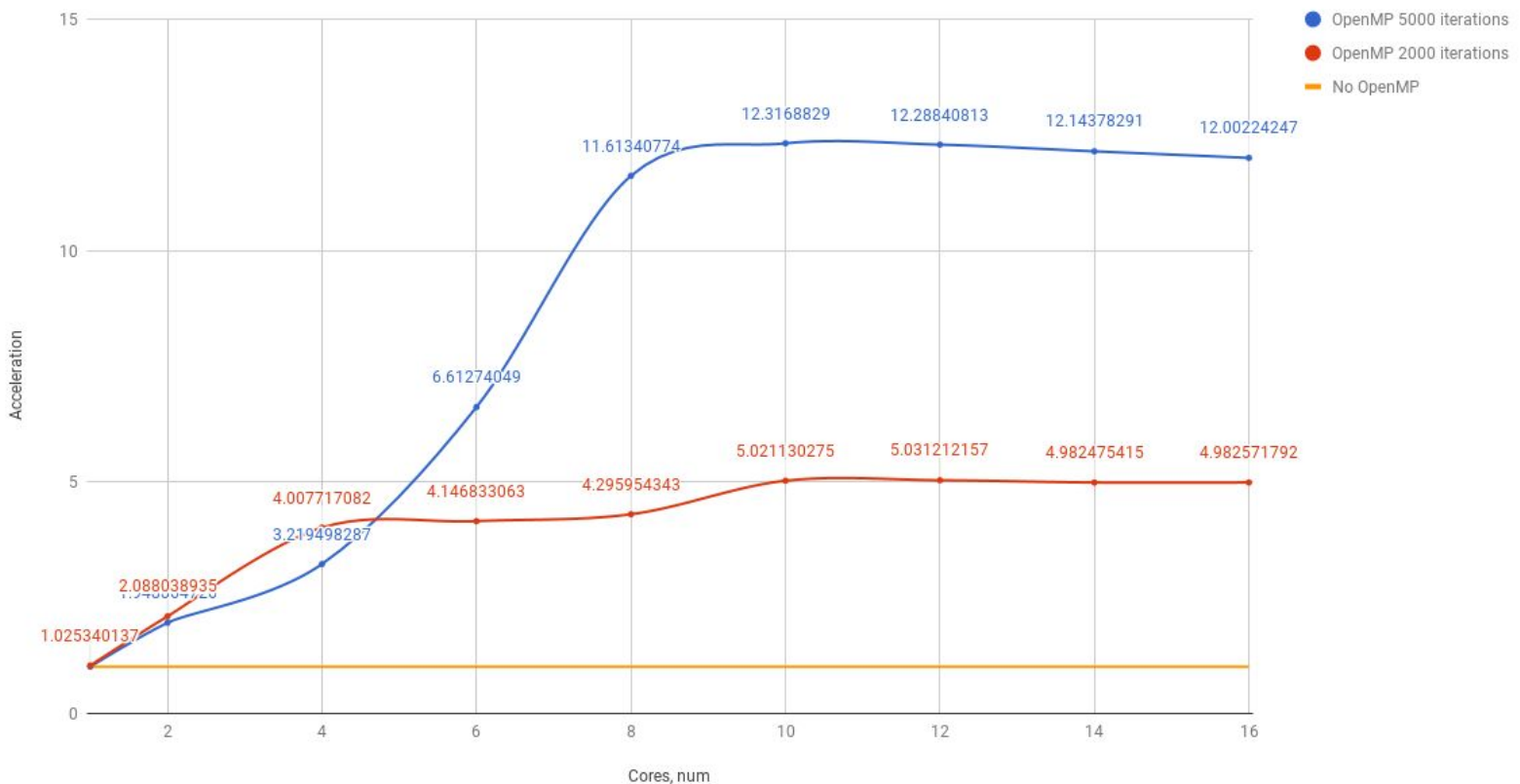
Графики

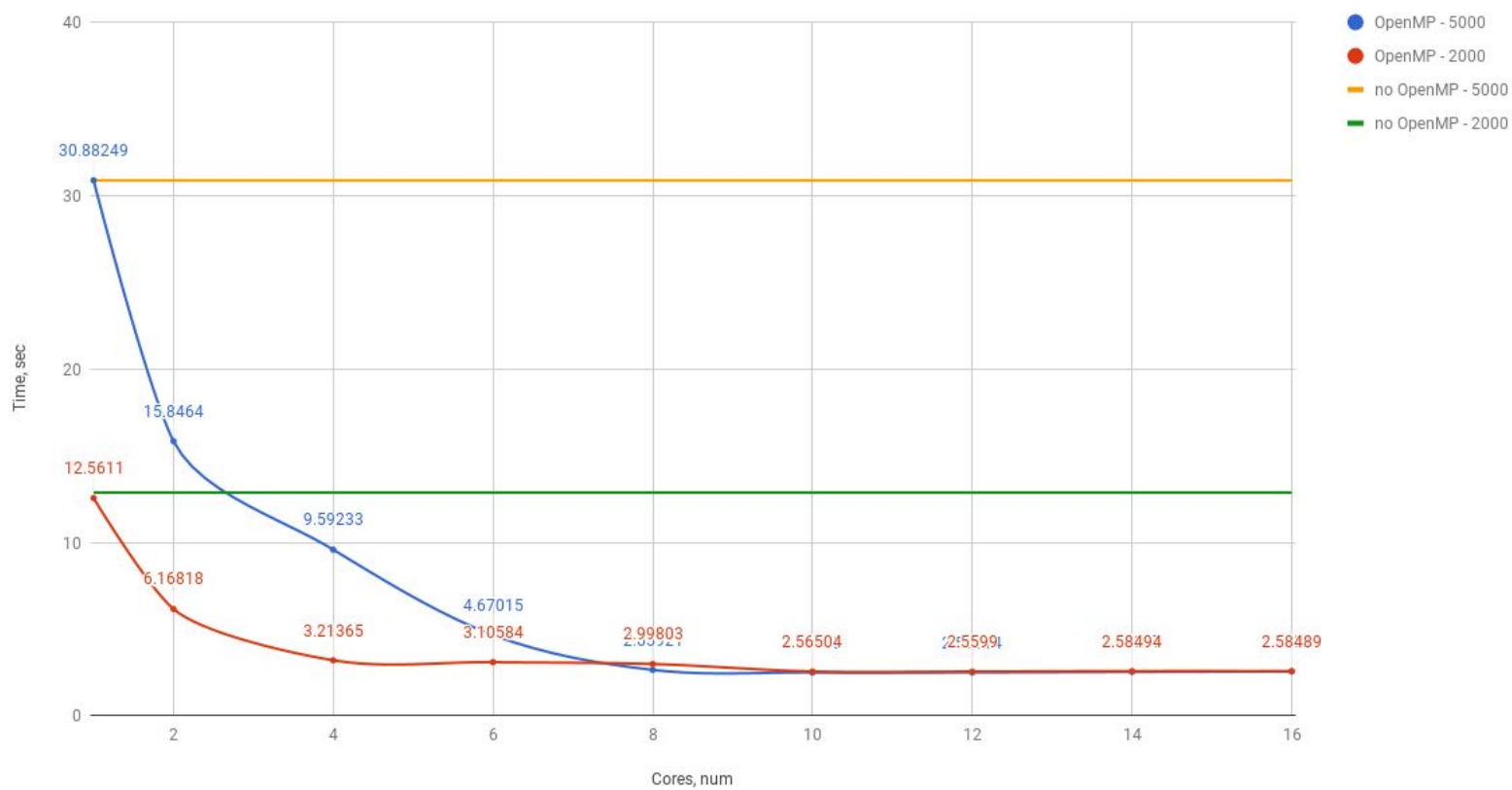
После подсчета времени выполнения на ск Regatta, выведем зависимости времени и ускорения от количества используемых в задаче процессоров на основе полученных данных.

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.