Производительность расчетов с использованием GPU в OpenFOAM

Александр Монаков
amonakov@ispras.ru

Владимир Платонов
soh@ispras.ru

Институт Системного Программирования

5 декабря 2013

GPU

- Специализированная массивно-параллельная архитектура
- 1-4 TFLOPS, 100-300 GB/s на одном устройстве
- Требуют задач с высоким параллелизмом

Преимущества

1. Производительность
2. Энергоэффективность (10x производительность при 2x энергопотреблении)
3. Доступность
   - В Top500: 53 системы с акселераторами, 39 NVIDIA
   - В персональных компьютерах

Использование в OpenFOAM

Часто основное время уходит на решение СЛАУ

- Метод сопряженных градиентов
- Многосеточный метод

Рассматриваем PCG

- Производительность ограничена:
  - Пропускной способностью памяти
  - Скоростью сети
- Сложности при переносе на GPU
  - Предобуславливание
  - Распараллеливание на несколько GPU
  - Минимизация накладных расходов

Предобуславливание на GPU

OpenFOAM: обычно используется DILU

- Подготовка/применение: решение треугольных систем
- Эффективный, но не обладает высоким параллелизмом
- Переупорядочивание может повысит паралелизм за счет снижения эффективности

Диагональное преобуславливание:

- Подготовка/применение: деление на диагональные коэффициенты
- Высокий параллелизм, низкая эффективность

Предобуславливание на GPU

Наш выбор для GPU: AINV

$$W^T AZ = D + e$$

- Подготовка: вычисление явного разложения приближенной обратной матрицы
- Применение: домножение на две разреженные матрицы
- Хорошо подходит для GPU
- Затраты на подготовку соизмеримы с решением СЛАУ

Асинхронное вычисление AINV

Предположим:

1. Что связность сетки не меняется
   $\Rightarrow$ Портрет разреженной матрицы СЛАУ фиксирован
2. СЛАУ для последних шагов по времени имеют близкие коэффициенты
   $\Rightarrow$ Можно переиспользовать предобуславливатели

Тогда:

1. Предобуславливатели вычисляются асинхронно в отдельном потоке
2. На GPU используется последний посчитанный предобуславливатель

Параллельное решение

1: Несколько MPI-процессов, один GPU

1. Собрать на мастер-процессе объединенную матрицу СЛАУ
2. Решить на GPU как в однопроцессном случае
3. Распределить найденный вектор-решение СЛАУ

Особенности:

- В цикле работы с GPU нет MPI-коммуникаций
- Более эффективное использование GPU

Параллельное решение

2: Несколько MPI-процессов, несколько GPU
Задача: позволить $N_{\text{GPUs}} < N_{\text{CPUs}}$

1. Объединить MPI-процессы по GPU
2. Собрать объединенные матрицы для GPU-групп
3. В цикле PCG возникают MPI-редукции
4. Необходимо учитывать граничные коэффициенты

Параллельное решение

2: Несколько MPI-процессов, несколько GPU
Задача: позволить $N_{\text{GPUs}} < N_{\text{CPUs}}$

1. Объединить MPI-процессы по GPU
2. Собрать объединенные матрицы для GPU-групп
3. В цикле PCG возникают MPI-редукции
4. Необходимо учитывать граничные коэффициенты

Снижение накладных расходов

1. Преобразование матрицы в GPU-формат только на первом шаге
   На последующих шагах – обновлять ненулевые коэффициенты

2. Использовать библиотеку MPI с поддержкой CUDA
   - MVAPICH2
   - OpenMPI

3. MPI-3: асинхронные редукции

Снижение накладных расходов

1. Преобразование матрицы в GPU-формат только на первом шаге
   На последующих шагах – обновлять ненулевые коэффициенты

2. Использовать библиотеку MPI с поддержкой CUDA
   - MVAPICH2
   - OpenMPI

3. MPI-3: асинхронные редукции

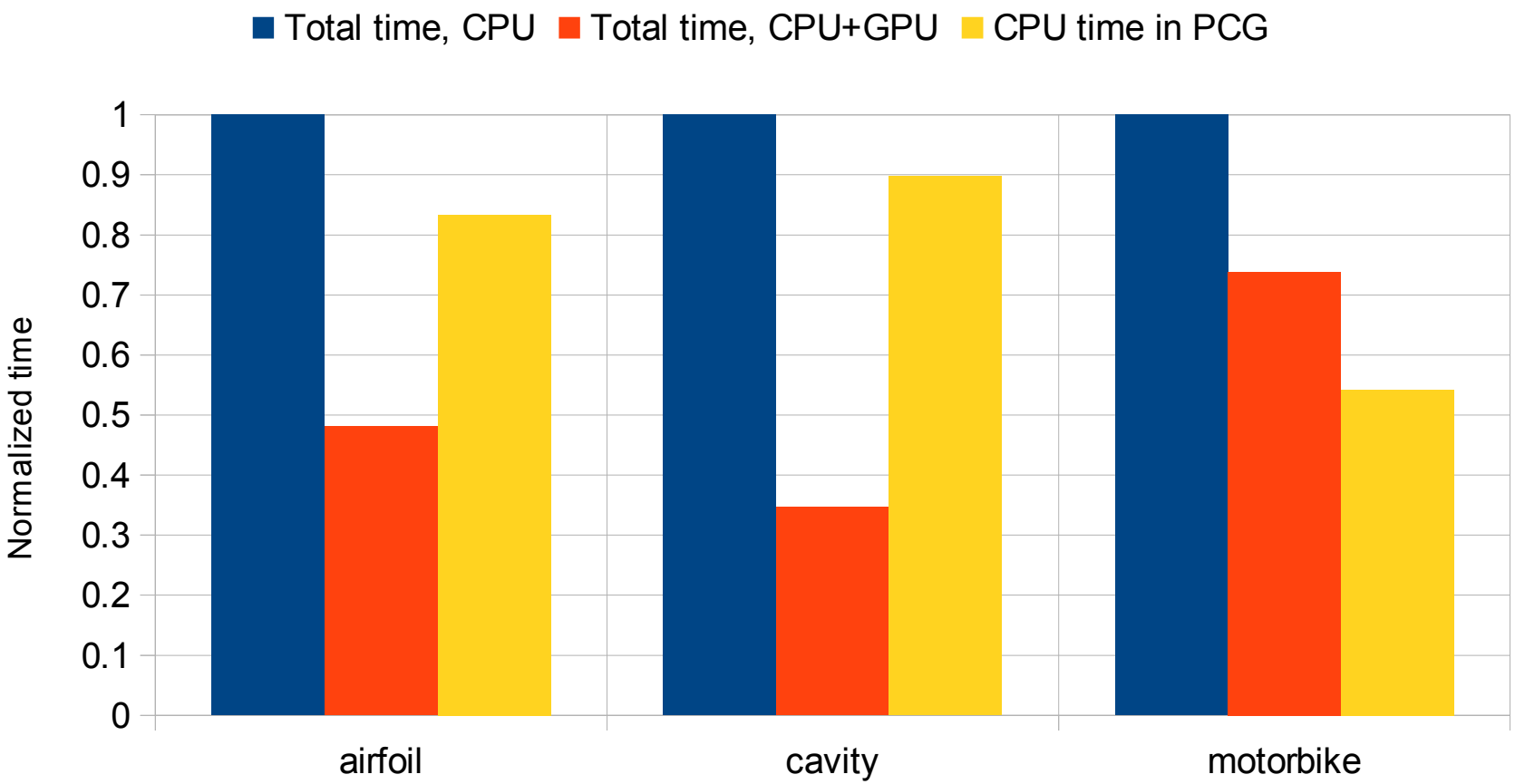
Снижение накладных расходов

1. Преобразование матрицы в GPU-формат только на первом шаге
   На последующих шагах – обновлять ненулевые коэффициенты

2. Использовать библиотеку MPI с поддержкой CUDA
   - MVAPICH2
   - OpenMPI

3. MPI-3: асинхронные редукции

Производительность на 1 GPU

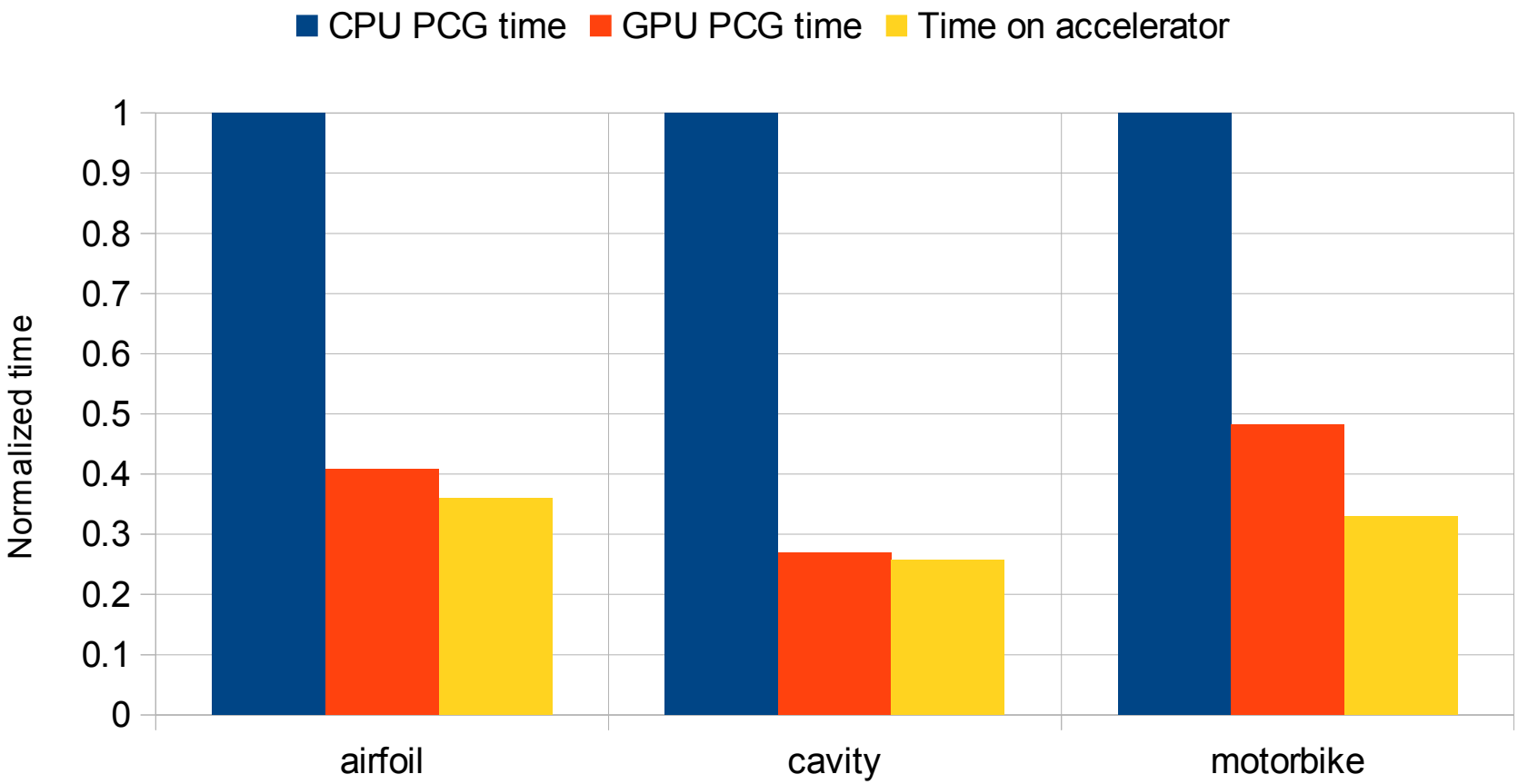


AIRFOIL, `pisoFoam`, 1.3M ячеек

CAVITY, `icoFoam`, 2M ячеек

MOTORBIKE, `simpleFoam`, 2.75M ячеек

CPU: 2x 6-core Intel Xeon, GPU: Tesla M2090


CPU PCG time
GPU PCG time
Time on accelerator
Normalized time
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
airfoil
cavity
motorbike

Case	$t_{\text{step}}^{\text{PCG}}$, ms	$t_{\text{startup}}^{\text{GPU}}$, ms	GPU Speedup	$N_{\text{iter}}^{=}$	$N_{\text{iter}}^{2\text{x}}$
AIRFOIL	14	150	2.8	17	75
CAVITY	30	130	3.9	6	18
MOTORBIKE	50	250	3.0	8	30

$t_{\text{step}}^{\text{PCG}}$: время на одну итерацию PCG

$t_{\text{overhead}}^{\text{GPU}}$: накладные расходы на копирование памяти CPU-GPU

$N_{\text{iter}}^{=}$: количество итераций, когда достигается паритет

$N_{\text{iter}}^{2\text{x}}$: количество итераций, когда GPU в 2 раза быстрее

Спасибо!