Реализация Multi-GPU версии OpenFOAM

Александр Монаков, amonakov@ispras.ru

Институт Системного Программирования

6 декабря 2012 г.

GPU

Массивно-параллельные вычислительные устройства

- Ресурсы производительности на порядок выше по сравнению с процессорами: $\approx$ 1–3 TFLOP/s, $\approx$ 100–250 GB/s
- Акселераторы используют собственную память, доступную для CPU через PCI-Express
- Существенный прирост общей производительности и сокращение энергозатрат для подходящих задач
- Как правило, требуют использование специализированных структур данных и алгоритмов для достижения оптимальной производительности

Перенос вычислений на GPU

- Распараллеливание в OpenFOAM достигается только за счёт разделения сетки между MPI-процессами
- Наиболее затратным является решение симметричных систем (50 – 80%)

Перенос метода сопряжённых градиентов на GPU:

- Работа с разреженной матрицей системы на GPU
- Предобуславливание
- Распараллеливание солвера

Разреженные матрицы на GPU

- Единственная операция, требующая оптимизации – умножение на плотный вектор
- Существенно зависит от выбора структуры данных
- Мы используем формат Sliced ELLPACK
  - Более высокая производительность по сравнению с библиотеками CUSPARSE, cusp
  - Требуют больше времени для преобразования формата

Разреженные матрицы на GPU

Как правило, портрет разреженной матрицы одинаков для всех решаемых систем

- Можно выполнить преобразование формата один раз
- Запомнить портрет и далее обновлять только ненулевые коэффициенты на GPU
- Важная оптимизация для сложных форматов

Предобуславливание на GPU

- Эффективное предобуславливание для GPU-акселераторов – активно исследуемая тема
- Преобуславливатели ILU-семейства не подходят для GPU:
  - Решение треугольных систем плохо параллелится
  - Трудности с векторизацией кода
- Удобно использовать предобуславливатели, основанные на построении приближённой обратной матрицы явно

AINV на GPU

Для реализации был выбран AINV, представляющий обратную матрицу в виде произведения треугольных:

$$W^T A Z = D + e$$

W, Z – верхнетреугольные с единицами на диагонали, $W = Z$ для симметричных
Построение параметризовано drop tolerance

- Применение на GPU – две операции умножения разреженной матрицы на вектор
- Оптимизация drop tolerance – открытый вопрос

Вычисление предобуславливателя

Временные затраты на вычисление AINV на CPU соизмеримы с временем решения системы на GPU

Мы используем оптимизацию, основанную на предположении, что последовательно решаемые системы имеют близкие коэффициенты, и преобуславливатели от ранее решённых систем хорошо подходят для новых

- Можно несколько раз переиспользовать предобуславливатель
- Можно полностью скрывать затраты, вычисляя его асинхронно и параллельно с решением системы на GPU

Оптимизации AINV, специфичные для GPU

- Хранить и применять предобуславливатель в одинарной точности (float)
- Переупорядочивание множителей в лево-верхнетреугольную форму

Использование нескольких GPU

- 1 узел, несколько MPI-процессов + 1 GPU
  - Промежуточный этап между последовательной и параллельной реализациями
  - Полезный режим для исследователей с компьютерами с GPU
- Несколько узлов, $N_{CPUs} > N_{GPUs}$

MPI + 1 GPU

- GPU используется мастер-процессом
- Без MPI-коммуникаций в солвере
- Решается объединённая матрица системы

Multi-GPU

Цель – эффективная реализация GPU-решателя для случая

$$N_{CPUs} \geq N_{GPUs}$$

. . . без одновременного использования GPU из разных процессов

- Количество MPI-процессов = количество CPU-ядер
- Каждый из доступных акселераторов используется лишь одним MPI-процессом

Преимущества схемы:

- Увеличение эффективности
- Хорошая совместимость

MPI-коммуникации в Multi-GPU

Использование CUDA+MPI в плагине:

1. GPU CUBLAS $\rightarrow$ «pinned»-память $\rightarrow$ MPI-редукция
2. GPU ядро $\leftrightarrow$ оперативная память $\leftrightarrow$ MPI Send/Recv

Простая схема, не требует поддержки CUDA в реализации MPI

Производительность

17 узлов: 2x 6-core Xeon 5650, 3x Tesla M2070
51 GPU, 204 ядра CPU
Сетка с 50M ячеек
Только ресурсы CPU:

1. 51 ядро: 2553 s
2. 216 ядер: 1314 s

С использованием GPU:

1. 51 ядро + 51 GPU: 633 s
2. 102 ядра + 51 GPU: 525 s
3. 204 ядра + 51 GPU: 531 s