Эффективное программирование современных микропроцессоров и мультипроцессоров

Практическое задание 3

Цель: научиться оптимизировать использование памяти в простых программах численного моделирования.

Теория

При решении вычислительных задач на сетках часто встречающейся операцией является обход всех элементов сетки для пересчёта некоторых сеточных значений. При этом сеточные значения считываются из оперативной памяти, участвуют в вычислениях, и некоторые из них записываются обратно в оперативную память. Если обращения в память не могут быть скрыты за вычислениями (если вычислительных операций недостаточно или из-за информационных зависимостей), то подсистема памяти становится узким местом.

Одним из способов решения проблемы является такая реорганизация вычислений, при которой за один обход элементов сетки выполняется сразу несколько последовательно идущих этапов вычислений. Таким образом, данные, будучи загруженными в регистры или кэш-память, участвуют в большем объёме вычислений, т.е. используются более эффективно. Соответственно, сами данные приходится загружать меньшее число раз. В результате арифметическая интенсивность (FLOPS/Byte) увеличивается.

Для примера рассмотрим оптимизацию алгоритма, состоящего из последовательности итераций, на каждой из которых по схеме «крест» пересчитываются все сеточные элементы, кроме граничных. Пусть исходный текст программы на псевдокоде выглядит следующим образом:

```
for (it=0;it<nt;it++) { // цикл по итерациям (по времени)
  for (i=1;i<ny-1;i++) // циклы по сетке (по пространству)
    for (j=1;j<nx-1;j++)
      v[i][j] = count(u[i][j],u[i-1][j],u[i+1][j],u[i][j-1],u[i][j+1],it);
  swap(u,v);
}
```

Допустим, только что были вычислены значения строк v[1][] и v[2][] для текущей итерации it. Заметим, что элементы этих и соседних строк обоих массивов сейчас лежат в кэш-памяти. Теперь можно уже вычислить строку u[1] для итерации it+1. Далее вычисляем значения строи v[3][] для итерации it. После этого уже можно вычислить значения u[2][] для итерации it+1. И так далее. Алгоритм, описывающий выполнение двух итераций за один проход, выглядит следующим образом:

```
for (it=0;it<nt;it+=2) { // цикл по итерациям (по времени)
  { i = 1;
    for (j=1;j<nx-1;j++)
      v[i][j] = count(u[i][j],u[i-1][j],u[i+1][j],u[i][j-1],u[i][j+1],it);
  }
  for (i=2;i<ny-1;i++) {
    for (j=1;j<nx-1;j++)
      v[i  ][j] = count(u[i  ][j],u[i-1][j],u[i+1][j],u[i  ][j-1],u[i  ][j+1],it  );
    for (j=1;j<nx-1;j++)
      u[i-1][j] = count(v[i-1][j],v[i-2][j],v[i  ][j],v[i-1][j-1],v[i-1][j+1],it+1);
  }
  { i = ny-1;
    for (j=1;j<nx-1;j++)
      u[i-1][j] = count(v[i-1][j],v[i-2][j],v[i  ][j],v[i-1][j-1],v[i-1][j+1],it+1);
  }
}
```

Постановка задачи

1. В векторизованной программе из практического задания 2 реорганизовать вычисления таким образом, чтобы за один проход по сетке выполнялось сразу несколько итераций метода. Сделать сначала реализацию для двух итераций за проход. Если есть ускорение, сделать реализацию для трёх итераций за проход. И так далее, пока ускорение не перестанет иметь место.
2. Проанализировать производительность наиболее быстрой версии программы аналогично анализу в задании 2 (включая roofline-модель). Сравнить результаты анализа с результатами в задании 2.

Отчёт

Отчёт по работе должен содержать:

- ФИО, группа, номер лабораторной работы, номер варианта
- Задание (коротко)
- Описание параметров тестирования:
    - полное название процессора,
    - название и версия компилятора, ключи компиляции,
    - параметры программы.
- Времена работы программы при выполнении различного числа итераций за один обход массива.
- Листинг самой быстрой версии программы (в приложении).
- Результаты анализа производительности и сравнения с программой из задания 2, включая roofline-модель.
- Вывод