Новосибирский государственный университет
Факультет информационных технологий
Кафедра параллельных вычислений

Эффективное программирование современных микропроцессоров и мультипроцессоров

Оптимизация ветвлений и циклов

Преподаватели:
Киреев С.Е.
Калгин К.В.

ВЕТВЛЕНИЯ

Ветвления

- Команды ветвления (передачи управления)
  - Безусловные / Условные
  - Прямые / Косвенные
- Где встречаются команды ветвления
  - Оператор goto
  - Вызов подпрограмм
  - Возврат из подпрограмм
  - Оператор if
  - Оператор “?:” (x = a>b ? y : z;)
  - Оператор switch
  - Операторы циклов: for, while

Ветвления

- Какие затраты на переход
  - Правильно предсказанный: 0-2 такта
  - Неправильно предсказанный: 12-25 тактов
- Что требуется предсказывать
  - Результат перехода (да/нет) – по таблице результатов
  - Направление перехода (адрес) – по таблице адресов
- Способ предсказания результата перехода
  - Статический
  - Динамический

Ветвления

- Предсказание направления перехода
  - Многократные переходы (прямые и косвенные) по одному и тому же адресу предсказываются хорошо
    - Адрес запоминается в BTB (Branch Target Buffer)
  - Многократные косвенные переходы по разным адресам предсказываются плохо

Ветвления

- Статическое предсказание результата перехода
  - Безусловные переходы
  - Условные переходы «вперёд» – не произойдут, «назад» – произойдут

Ветвления

- Динамическое предсказание результата перехода – какие параметры важны:
  - Локальный предсказатель
    - Длина локальной истории переходов
  - Глобальный предсказатель
    - Длина глобальной истории переходов
  - Предсказатель циклов
    - Максимальное число итераций цикла
  - Предсказатель возвратов из подпрограмм
    - Парные команды call/ret

Ветвления

- Предсказание ветвлений в циклах
  - Хорошо предсказываются команды перехода, история срабатывания которых образуют периодический шаблон с небольшим периодом: 0000000, 1111111, 0101010101, 000100010001, 011000101100011011000 1
  - Худший случай – равновероятный случайный исход

- Оптимизация ветвлений
  - Устранение ветвления
    - Замена на условное присваивание: cmov
    - Замена на выбор из таблицы
  - Вынос ветвления за пределы цикла (loop unswitching – размыкание цикла)

Ветвления

- Реализация switch
  - Таблица переходов
    - Одна команда косвенного перехода
    - Исход предсказывается хорошо
    - Адрес перехода может предсказываться плохо
  - Дерево ветвлений
    - Направление перехода может предсказываться плохо
    - Адрес перехода предсказывается хорошо
- Switch хорошо предсказывается, если его выбор повторяется
- Оптимизация
  - наиболее часто выбираемый вариант вынести в отдельный if
  - остальные варианты отсортировать в порядке убывания вероятностей

Ветвления

- Использование подпрограмм – затраты:
  - Вызов и возврат занимает время
  - Передача параметров, сохранение регистров, настройка стека занимает время и ресурсы
  - Адрес функции и возврата занимают запись в таблице адресов перехода
  - Если код сильно фрагментирован и разбросан, кэш команд работает хуже
  - Вызов виртуальных методов класса (косвенный переход) может плохо предсказываться

Ветвления

- Оптимизация вызовов подпрограмм
  - Подстановка функций в код (inline, макросы)
  - Передача параметров через регистры
  - Использовать парные call/ret
  - Не использовать виртуальные методы
  - В каждом конкретном вызове виртуального метода обращаться только к одной конкретной функции
  - В критических циклах не делать вызовы функций

Ветвления

- Если в критическом участке кода много ветвлений, то они могут плохо предсказываться из-за конфликтов в таблице адресов
  - Даже безусловные переходы и call/ret

ПРЕОБРАЗОВАНИЯ ЦИКЛОВ

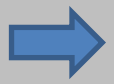
Преобразования циклов

- Расщепление тела цикла (loop distribution/fission)
- Слияние циклов (loop fusion/jamming)
- Расщепление цикла (loop splitting)
- Разгрузка циклов (loop peeling)
- Раскрутка цикла (loop unrolling)
- Перестановка циклов (loop interchanging)
- Изменение порядка итераций (loop reversal)
- Loop tiling
- Программная конвейеризация (software pipelining)

Преобразования циклов

- Расщепление тела цикла (loop distribution/fission)

```
for (i=1; i<N; i++)
{
  a[i] = b[i];
  c[i] = c[i-1] + 1;
}
```



```
for (i=1; i<N; i++)
  a[i] = b[i];

for (i=1; i<N; i++)
  c[i] = c[i-1] + 1;
```

Зачем:

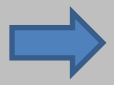
- Упрощение тела цикла, что позволяет в дальнейшем выполнить другие преобразования
- Улучшение локальности обращений за счёт обращения к меньшему количеству данных
- Уменьшение числа обходимых массивов для аппаратной предвыборки

Преобразования циклов

- Слияние цикла (loop fusion/jamming)

```
for (i=0; i<N; i++)
  a[i] = 0;

for (i=0; i<N; i++)
  b[i] = 1;
```



```
for (i=0; i<N; i++)
{
  a[i] = 0;
  b[i] = 1;
}
```

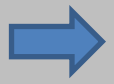
Зачем:

- Уменьшение накладных расходов на организацию цикла
- Увеличение объема работы, выполняемой в цикле (чтобы его имело смысл распараллеливать)
- Улучшение локальности обращений за счёт комбинирования обращений к одним и тем же данным

Преобразования циклов

- Расщепление цикла (loop splitting) – разбиение пространства итераций

```
int p = 10;
for (int i=0; i<10; ++i)
{
  y[i] = x[i] + x[p];
  p = i;
}
```



```
y[0] = x[0] + x[10];
for (int i=1; i<10; ++i)
{
  y[i] = x[i] + x[i-1];
}
```

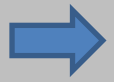
Зачем:

- Упрощение тела цикла
- Устранение зависимостей в цикле

Преобразования циклов

- Разгрузка циклов (loop peeling)
  - выделение первых/последних итераций

```
for (i=0; i<N; i++)
  a[i] = b[i] + c[i];
```



```
a[0] = b[0] + c[0];
for (i=1; i<N; i++)
  a[i] = b[i] + c[i];
```

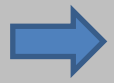
Зачем:

- Используется для обеспечения выровненного доступа к элементам массива перед векторизацией

Преобразования циклов

- Раскрутка цикла (loop unrolling)

```
s = 0;
for (i=0; i<N; i++)
  s += a[i];
```



```
s = 0;
for (i=0; i<N; i+=4) {
  s += a[i];
  s += a[i+1];
  s += a[i+2];
  s += a[i+3];
}
```

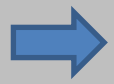
Зачем:

- + Увеличение степени параллелизма команд
- + Уменьшение накладных расходов на организацию цикла
- - Увеличивается «давление на регистры»
- - Код имеет больший размер

Преобразования циклов

- Раскрутка внешнего цикла с последующим слиянием внутренних

```
for (i=0; i<N; i++)
  for (k=0; k<N; k++)
    for (j=0; j<N; j++)
      c[i][j] += a[i][k]*b[k][j];
```



```
for (i=0; i<N; i+=2)
  for (k=0; k<N; k++)
    for (j=0; j<N; j++) {
      c[i][j] += a[i][k]*b[k][j];
      c[i+1][j] += a[i+1][k]*b[k][j];
}
```

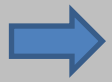
Зачем:

- Улучшение локальности обращений за счёт комбинирования обращений к одним и тем же данным

Преобразования циклов

- Перестановка циклов (loop interchanging)

```
for (j=1; j<N; j++)
  for (i=2; i<N; i++)
    a[i][j] = a[i-1][j] + b[i];
```



```
for (i=2; i<N; i++)
  for (j=1; j<N; j++)
    a[i][j] = a[i-1][j] + b[i];
```

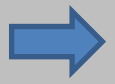
Зачем:

- Улучшение локальности обращений к памяти, организации последовательного обхода
- Подготовка кода для других оптимизаций (например, для векторизации)

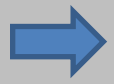
Преобразования циклов

- Изменение порядка итераций (loop reversal)

```
for (i=0;i<N;i++) {
  a[i] = b[i] + 1;
  c[i] = a[i] / 2;
}
for (j=0;j<N;j++)
  d[j] += 1/c[j+1];
```



```
for (i=N-1;i>=0;i--) {
  a[i] = b[i] + 1;
  c[i] = a[i] / 2;
}
for (j=N-1;j>=0;j--)
  d[j] += 1/c[j+1];
```



```
for (i=N-1;i>=0;i--) {
  a[i] = b[i] + 1;
  c[i] = a[i] / 2;
  d[j] += 1/c[j+1];
}
```

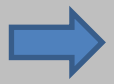
Зачем:

- Подготовка для других оптимизаций

Преобразования циклов

- Loop tiling – разбиение пространства итераций цикла на блоки

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    c[i] = c[i] + a[i, j] * b[j];
```



```
for (i = 0; i < N; i += 2)
  for (j = 0; j < N; j += 2)
    for (ii = i; ii < min(i + 2, N); ii++)
      for (jj = j; jj < min(j + 2,N); jj++)
        c[ii] = c[ii] + a[ii, jj] * b[ii];
```

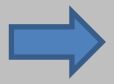
Зачем:

- для более эффективного использования кэш-памяти

Преобразования циклов

- Программная конвейеризация цикла (software pipelining)

```
for (i=0;i<N;i++) {
  x = a[i];
  y = x * 10;
  b[i] = y;
}
```



```
x1 = a[0]; y3 = x1*10; x2 = a[1];
for (i=2;i<N;i++) {
  x1 = a[i];
  y2 = x2 * 10;
  b[i-2] = y3;
  y3=y2; x2=x1;
}
y2 = x2*10; b[N-2] = y3;
b[N-1] = y2;
```

Зачем:

- для выявления параллелизма между операциями внутри одной итерации цикла