Оптимизация и измерение производительности

Судаков А.А.

“Параллельные и распределенные вычисления” Лекция 23

План

- Оптимизация
- Профилировка
- Измерение производительности

Литература

- http://vision.eng.shu.ac.uk/bala/c/c/optimisation/1/optimization.html
- http://www.top500.org/lists/linpack.php
- http://www.pallas.com/e/products/index.htm

Необходимость оптимизации

- Оптимизация необходима
  - Ускорение работы программы
- Оптимизировать необходимо только после того, как программа отлажена !!!
- Не отлаженную программу оптимизировать категорически не рекомендуется
- Для некоторых компиляторов и некоторых программ оптимизация может привести и неправильной работе программы

Подходы к оптимизации

- Оптимизация исполняемого кода
  - Использование специфических команд процессора
    - Векторные операции
  - Развертывание циклов
  - Изменение порядка следования инструкций
- Оптимизация исходного кода
  - Устранение повторяющихся операций
  - Оптимизация приема передачи данных
  - Оптимизация распараллеливания

Основные принципы

- Необходимо понимать, что делает алгоритм и как он это делает
- Оптимизировать необходимо
  - Самые медленные участки программы
  - Самые часто повторяющиеся участки программы
- Функцию `f` необходимо оптимизировать

```
for(i = 0; i<100; i++) f(i);
```

- Необходимо оптимизировать внутренний цикл

```
for(i = 0; i<100; i++)
for(j = 0; j<100; i++) …
```

Устранение ненужных участков

- if(x != 0) x=0;

Оптимизация за счет кэширования данных

- Обрабатывать данные небольшими блоками
- К массивам данных обращаться последовательно

Обработка данных блоками

- Большие блоки данных могут не помещаться в кэш процессора
  - float[10][10]
  - float[100][100] - будет обрабатываться не в 100 дольше, а более медленно
- Желательно, чтобы размеры структур и буферов соответствовали размеру строки кэша

Обращение к данным

- $C_{ij}=A_{ik}B_{kj}$

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

- $C_{ij}=A_{ik}B_{kj}$

```
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]
```

Самый внутренний индекс должен быть самым левым (для С) или самым правым (для фортрана)

Уменьшение количества вызовов функций (inline)

```
int foo(a, b) {
  a = a - b;
  b++;
  a = a * b;
  return a;
 }
```

Быстрее

```
#define foo(a, b) (((a)-(b)) * ((b)+1))
```

Развертывание циклов (unroll)

```
for (i = 0; i < 100; i++) {
      do_stuff(i);
}

for (i = 0; i < 100; ) {
      do_stuff(i); i++;
      do_stuff(i); i++;
      do_stuff(i); i++;
      do_stuff(i); i++;
      do_stuff(i); i++;
      do_stuff(i); i++;
      do_stuff(i); i++;
      do_stuff(i); i++;
       do_stuff(i); i++;
      do_stuff(i); i++;
}
```

Устранение ненужных циклов (loop jump)

```
for (i = 0; i < MAX; i++)
/* initialize 2d array to 0's */
 for (j = 0; j < MAX; j++)
   a[i][j] = 0.0; for (i = 0; i < MAX; i++)
    /* put 1's along the diagonal */
     a[i][i] = 1.0;

for (i = 0; i < MAX; i++) {
   for (j = 0; j < MAX; j++) {
     /* initialize 2d array to 0's */
    a[i][j] = 0.0;
   }
    /* put 1's along the diagonal */
   a[i][i] = 1.0;
}
```

Использование более быстрых операций (strength reduce)

```
x = w % 8;
y = pow(x, 2.0);
z = y * 33;
for (i = 0; i < MAX; i++) {
     h = 14 * i;
     printf("%d", h);
 }

x = w & 7;            /* bit-and cheaper than remainder */
y = x * x;            /* mult is cheaper than power-of */
z = (y << 5) + y;          /* shift & add cheaper than mult */
for (i = h = 0; i < MAX; i++) {
     printf("%d", h);
     h += 14;         /* addition cheaper than mult */
 }
```

Замена вычислений табличными операциями

- Вместо того, чтобы вычислять функции использовать вычисленные заранее значения

Ближе к степени двойки

- Не стоит создавать массивы данных и другие структуры с размерами отличающимися от степени двойки
- Структуры должны быть по возможности меньшего размера (правильная упаковка)
- Часто динамическое выделение памяти получается быстрее статического
  - Размер порций данных уравнивается по границе строки кэша

Пример упаковки

```
/* sizeof = 64 bytes */
struct foo {
      float a;
      double b;
      float c;
      double d;
      short e;
      long f;
      short g;
      long h;
      char i;
      int j;
      char k;
      int l;
      };
```

```
/* sizeof = 48 bytes */
struct foo {
      double b;
      double d;
      long f;
       long h;
      float a;
      float c;
      int j;
      int l;
      short e;
      short g;
      char i;
      char k;
};
```

Опции компилятора

- Каждый компилятор имеет свои опции оптимизации
  - Gcc
  - Icc
  - G77
  - Ifc

Ввод-вывод

- Уменьшать время передачи
  - Передавать данные реже и большими порциями, а не чаще и маленькими
- Использовать асинхронный и не блокирующий ввод-вывод
- Использовать специальные опции протокола передачи данных для уменьшения задержек
  - TCP_NODELAY
- Использовать настройки протокола для уменьшения времени задержки
  - Procfs
- Уменьшать количество ненужных операций ввода-вывода

Параллельные программы

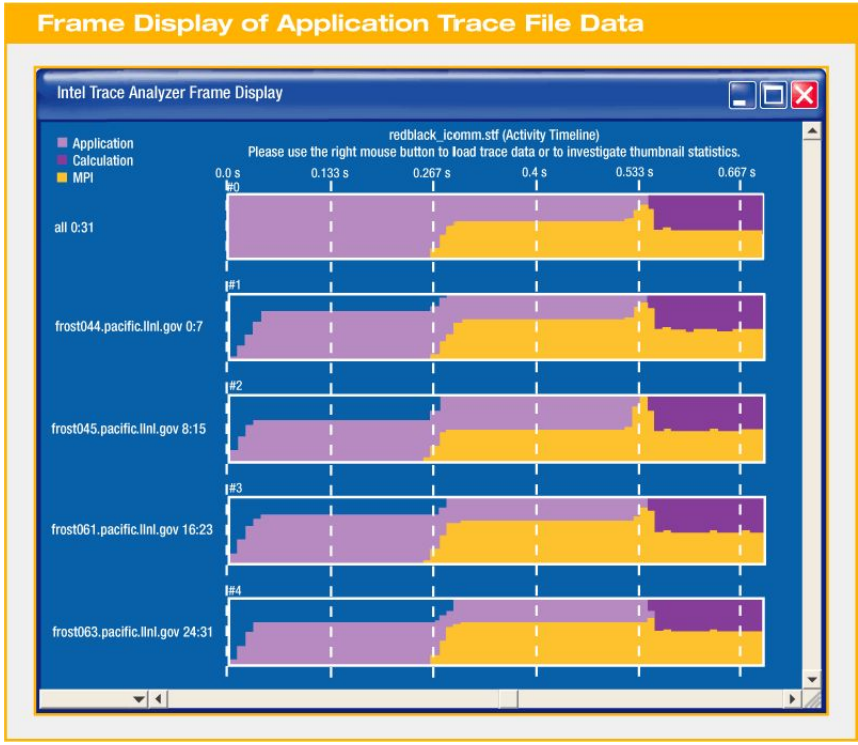
- Избегать гетерогенных машин
- Уменьшать количество последовательных операций, особенно при передаче данных
- Увеличивать гранулярность задач
- Использовать асинхронные операции передачи данных
  - Send ahead
- Лучше использовать конвейерные методы, особенно для кластеров и гетерогенных систем
- Для гетерогенной системы самые медленные машины должны быть самыми последними в цепочке

Профилирование

- Компилируется специальная информация для отслеживания времени выполнения каждой функции
- Специальные утилиты
  - Опции компилятора
  - Профилировщики
  - Специальные библиотеки

Примеры профилировщиков

- Gprof – для gcc
- Vampir
  - Visual MPI Resources



Измерение производительности

- Стандартные тесты
  - HPL high performance linpack benchmark
  - Bonnie benchmark http://linux.maruhn.com/sec/bonnie.html
  - Измерение производительности сети http://www.netperf.org/netperf/NetperfPage.html
  - MPI benchmarks
  - http://parallel.ru/
  - Тесты прикладных программ

HPL – используется в top5000

- Решение системы линейных уравнений методом Гаусса на параллельной машине
- Пользователи компилируют как угодно
- Во входном файле указываются параметры
  - Размеры блоков
  - Алгоритмы обмена
- Максимальный полученный результат отправляется на top500

Вопросы?