Оптимизация программ

Алексей А. Романенко
arom@ccfit.nsu.ru

О чем эта лекция?

Из лекции вы узнаете о:

- Типах оптимизации
- Подходах к оптимизации программ
- Инструментах профилирования

Оптимизация

Optimization consists of analyzing and tuning software code to make it perform faster and more efficiently on Intel processor architecture.

www.intel.com/products/glossary/body.htm

Для ЛЮБОЙ архитектуры!

Типы отимизаций

- По производительности
- По памяти
- По масштабируемости
- Оптимизация коммуникаций
- пр.

- Далее занимаемся только оптимизацией по производительности

Утверждения

- Дональд Кнут
  - Преждевременная оптимизация является корнем всего зла
- Если ваш код уже работает, его оптимизация верный путь к внедрению новых, и, возможно, более тонких ошибок
- Оптимизация делает код сложнее для понимания и поддержки
- Некоторые методы оптимизации по увеличению скорости снижают расширяемость код
- На оптимизацию можно потратить много времени и получит небольшой выигрыш в производительности. Кроме того это может привести к запутыванию кода
- Если вы слишком одержим оптимизации кода, люди будут называть тебя ботаник за вашей спиной

Утверждения

- Быстрые программы требуют больше памяти
- Чтобы создать более быструю программу требуется больше времени для оптимизации.
- Оптимизация кода для одной платформы может фактически сделать его хуже, на другой платформе

НО !!!!!

- Майкл Абраш (Quake, Unreal Tournament, Space Strike, Half-Life и т.д.):
- Производительность должна всегда измеряется
- Любая оптимизация, заметная пользователем пользователь, должна быть сделана.

Могут ли компиляторы выполнить оптимизацию?

- Могут. Но...
  - Компилятор не обладает базой данных алгоритмов
  - Компилятор не знает ничего о предметной области, в которой решается ваша задача
  - Компилятор не может полностью охватить весь ваш проект
  - Плохую реализацию не сможет поправить ни один компилятор
  - А может программа и должна работать так, как вы ее написали?

Стадии оптимизации

- Дизайн программы
  - Выбор алгоритмов и структур данных
  - Ускорение – 100 ...1000+
- Реализация программы
  - Языки программирования
  - Ускорение – 10 ...100+
- Профилирование программы
  - «Заточка» программы под архитектуру
  - Ускорение – 1 ...10+ (не для CELL BE)

Пример

```
s, i, N - integer
for (s=0, i=1; i<=N; i++)
     s += N/i;
```

Если N равно 40.000.000.000, то вычисление требует

- **~ 6 часов на Intel PIII-933!**
- **~ 1 час на Xeon Dualcore 2.4GHz (OpenMP, Pthreads)**

- **Давайте возьмем Quadro-core CPU и 4 CPU.**

Решение

N = 10

I	1	2	3	4	5	6	7	8	9	10
k=[N/I]	10	5	3	2	2	1	1	1	1	1

```
//[N/k] – индекс правой границы
//[N/(k+1)] – индекс левой границы

i = N;
while(i>0){
    //m – количество элементов с одинаковыми значениями
    m = i - N/(N/i + 1);
    s +=(N/i) * m;
    i -= m;
}
```

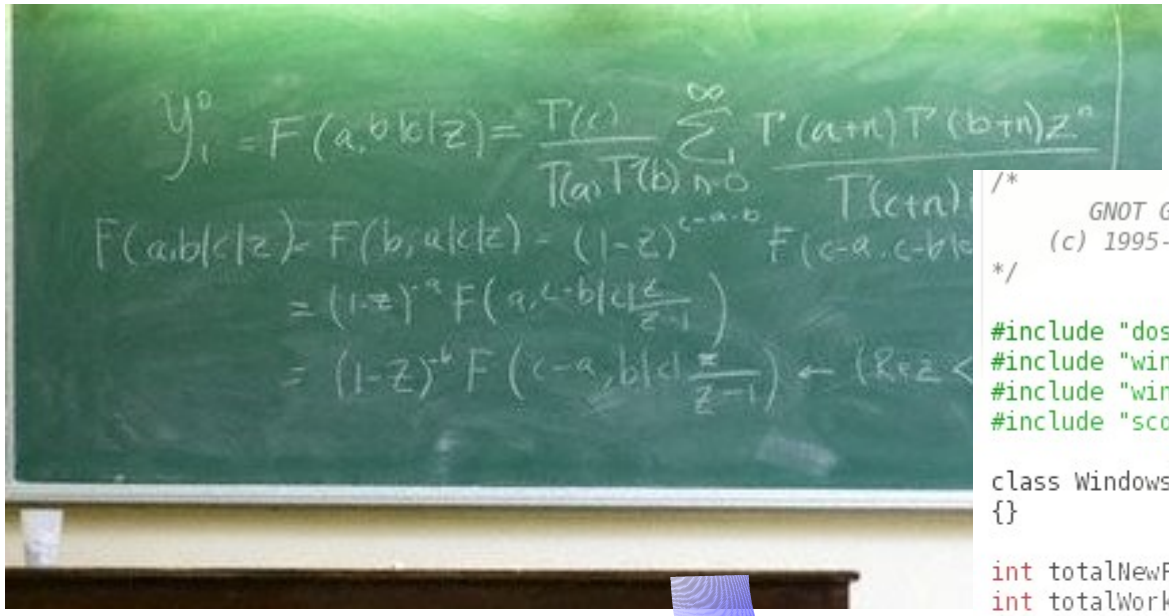
Меньше 1 сек!

Выбор алгоритма

- Читайте книги
  - The Art of Computer Programming, vol.1. Fundamental Algorithms by Donald E. Knuth
  - A. V. Aho, J. E. Hopcroft, and J. D. Ullman. The Design and Analysis of Computer Algorithms.
  - Работы в предметной области.
- Ищите симетрию
- Учитывайти граничные условия
- Применяйте воображение.

Если вашей программе суждено выполниться один раз, может быть, вам не нужно тратить время по ее оптимизации. Было бы лучше потратить это время на общение с друзья, когда программа работает.

Реализация программ



```
/*
        GNOT General Public License!
    (c) 1995-2007 Microsoft Corporation
*/

#include "dos.h"
#include "win95.h"
#include "win98.h"
#include "sco_unix.h"

class WindowsVista extends WindowsXP implements Nothing
{}

int totalNewFeatures = 3;
int totalWorkingNewFeatures = 0;
float numberOfBugs = 345889E+08;
boolean readyForRelease = FALSE;

void main {
    while (!CRASHED) {

    if (first_time_install) {
        if ((installedRAM < 2GB) ||
            (processorSpeed < 4GHz))
            {
                MessageBox("Hardware incompatible");
                GetKeyPress();
                ESCD();
            }
        }
        MakeLOGBs(apfile);
```

Реализация программ

- Циклы
- В\В с памятью
- Вызов функции
- Передача параметров
- пр.

Циклы

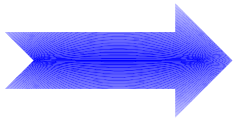
- Правило "10-90" - 90% времени программа проводит в 10% кода.
- Как правило, это 10% кода - циклы.
- Оптимизируя циклы можно значительно ускорить программу

Оптимизация циклов

- Вынос условных переходов за пределы циклов
- Исключить перевычисление
- Держать данные в кэше
- Разворачивать циклы (в случае простых циклов)

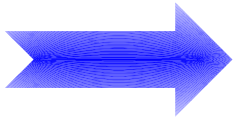
Условия внутри цикла

```
for(i=0; i<N; i++){
   if(i<N/2)
      // - foo
   else
      // - bar
}
```



```
for(i=0; i<N/2; i++){
   // - foo
}
for(i=N/2; i<N; i++){
   // - bar
}
```

```
for(i=0; i<N; i++){
   if(a == b)
      // - foo
   else
      // - bar
}
```



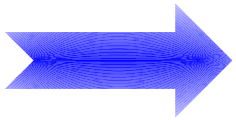
```
if(a == b)
   for(i=0; i<N; i++)
      // - foo
else
   for(i=0; i<N; i++)
      // - bar
```

Условия внутри цикла

- «Лишние» операции
  - сравнения
  - переходы
- Срыв конвейера

Перевычисление данных

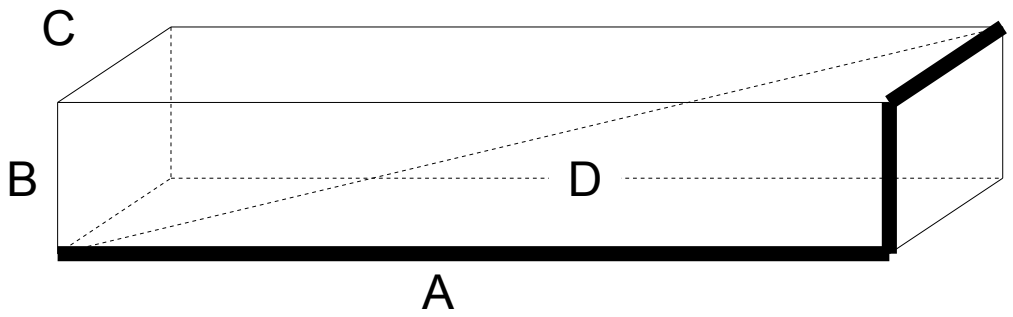
```
for(i=0; i<N; i++){
   s += F(i)*dx;
}
```



```
for(i=0; i<N; i++){
   s += F(i);
}
s = s*dx;
```

Задача

- Дан параллелепипед
- Грани A, B, C – целое
- 1 <= A, B, C, <=250
- Найти такой параллелепипед, в котором диагональ D - целое, D<=250



Решение 1

```
for(D=1; D<=250; D++){
   for(A=1; A<=250; A++){
      for(B=1; B<=250; B++){
         for(C=1; C<=250; C++){
            if(D*D*D ==  A*A*A + B*B*B + C*C*C){
               my_print(A, B, C, D);
            }
         }
      }
   }
}
```

Заметим

- Кубы можно вычислить заранее;
- Если известен оди параллелепипед (ABC), мы знаем другие 5: ACB, BAC, BCA, CAB, CBA. Изменяем функцию my_print;
- Можно проверить только грани: A<=B<=C

Решение 2

```
for(D=1; D<=250; D++) cube[D] = D*D*D;
for(D=1; D<=250; D++){
   for(A=1; A<D; A++){
      for(B=A; B<=250; B++){
         tmp = cube[D] - cube[A] - cube[B];
         if(tmp <= 0) break;
         for(C=B; C<=250; C++){
            if(tmp < cube[C]) break;
            if(tmp == cube[C]){
               my_print(A, B, C, D);
               break;
            }
         }
      }
   }
}
```

Дважды быстрее!

Держать данные в кэше

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

```
do I=1, N
   do J=1, K
      c(I,J) = a(I,J)*f(J)
   end do
end do
```

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

```
do J=1, K
   do I=1, N
      c(I,J) = a(I,J)*f(J)
   end do
end do
```

Элементы массивов

- Fortran:
  - m(1, 1), m(2, 1), m(3, 1), ... m(1, 2), m(2, 2), ...
- C/C++ :
  - m(0, 0), m(0, 1), m(0, 2), ... m(1, 0), m(1, 1), ...

- Данные в кэше лежат в кэш линиях
- Кыш линии от 8 до 512 байт
- Выбор данных из кэша существенно быстрее чем из памяти

Держать данных в кэше

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

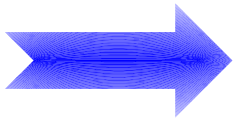
```
do I=1, N
   do J=1, K
      c(I,J) = a(I,J)*f(J)
   end do
end do
```

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

```
do J=1, K
   do I=1, N
      c(I,J) = a(I,J)*f(J)
   end do
end do
```

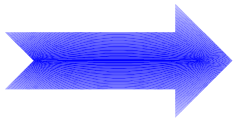
Простейшие операции

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



```
for(i=0; i<K; i++){
   a[i] = b[i]+c[i];
   d[i] = k[i]*f;
}
for(i=K; i<N; i++)
   d[i] = k[i]*f;
```

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



```
for(i=0; i<K/4; i+=4){
   a[i+0] = b[i+0]+c[i+0];
   a[i+1] = b[i+1]+c[i+1];
   a[i+2] = b[i+2]+c[i+2];
   a[i+3] = b[i+3]+c[i+3];
}
```

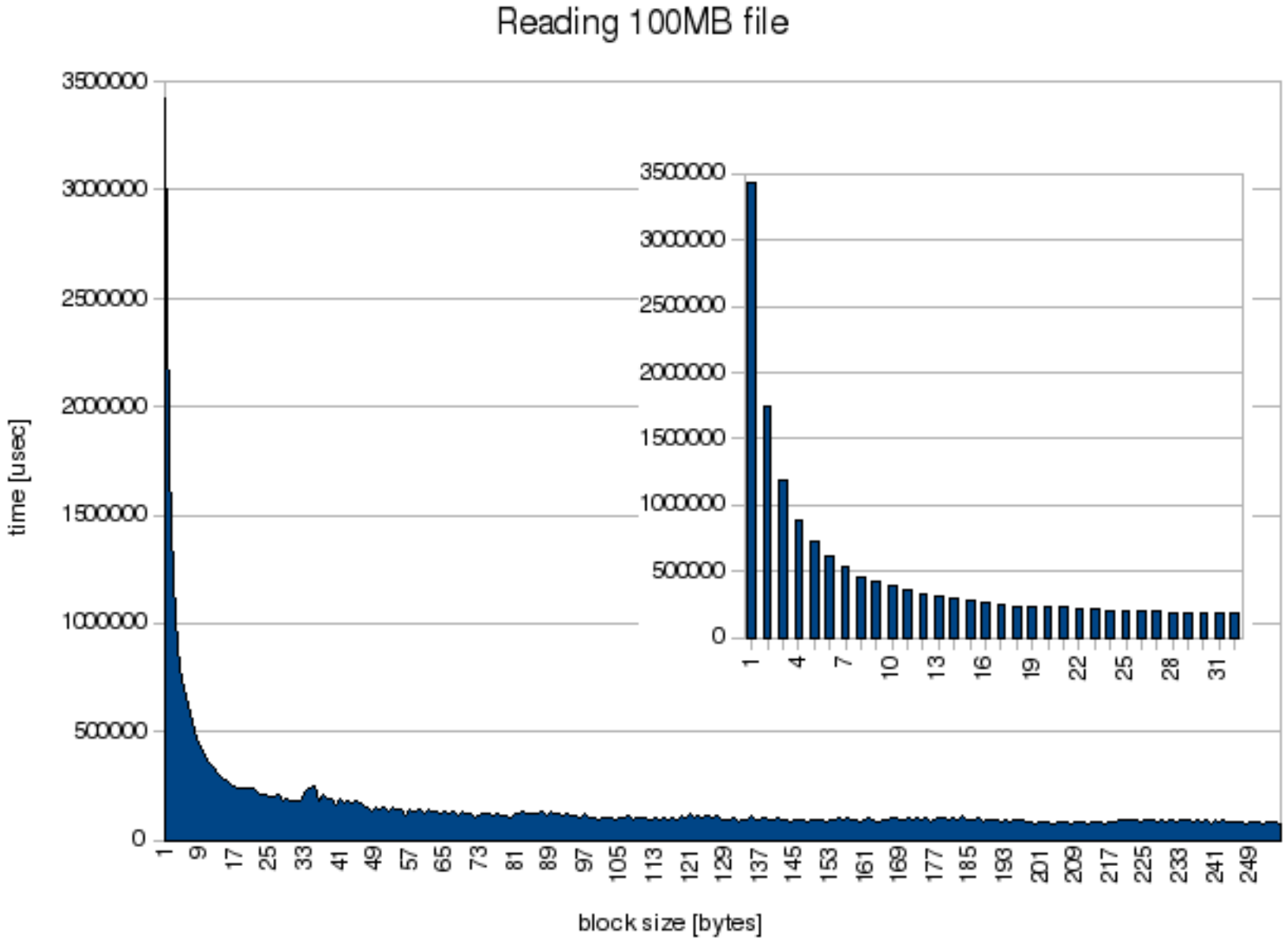
Операции файлового В\В

Чтение данных

```
char NextChar(FILE *fd){
   char ch;
   fread(&ch, 1, sizeof(char), fd);
   return ch;
}
```

Не читайте файл побайтово

Диаграмма чтения данных



Чтение данных

```
inline char NextChar1(FILE *fd){
   static char buf[64];
   static int ptr=0;
   static int tch=0;

   if(ptr==tch){
      ptr = 0;
      tch = fread(buf, sizeof(char), 64, fd);
   }
   return buf[ptr++];
}
```

- На Unix-like системах есть функции отображения файла на память
  - mmap, munmap

Запись данных

- Использование буферизованного вывода
- Не пишите много на консоль

```
for(i=0; i<N; i++){
    //do something
    printf("\rComplete %d%%", i*100/N);
}
```

Асинхронные операции чтения\записи

- int aio_cancel(int, struct aiocb *);
- int aio_error(const struct aiocb *);
- int aio_fsync(int, struct aiocb *);
- int aio_read(struct aiocb *);
- ssize_t aio_return(struct aiocb *);
- int aio_suspend(const struct aiocb *const[], int,
  const struct timespec *);
- int aio_write(struct aiocb *);

Двойной буфер

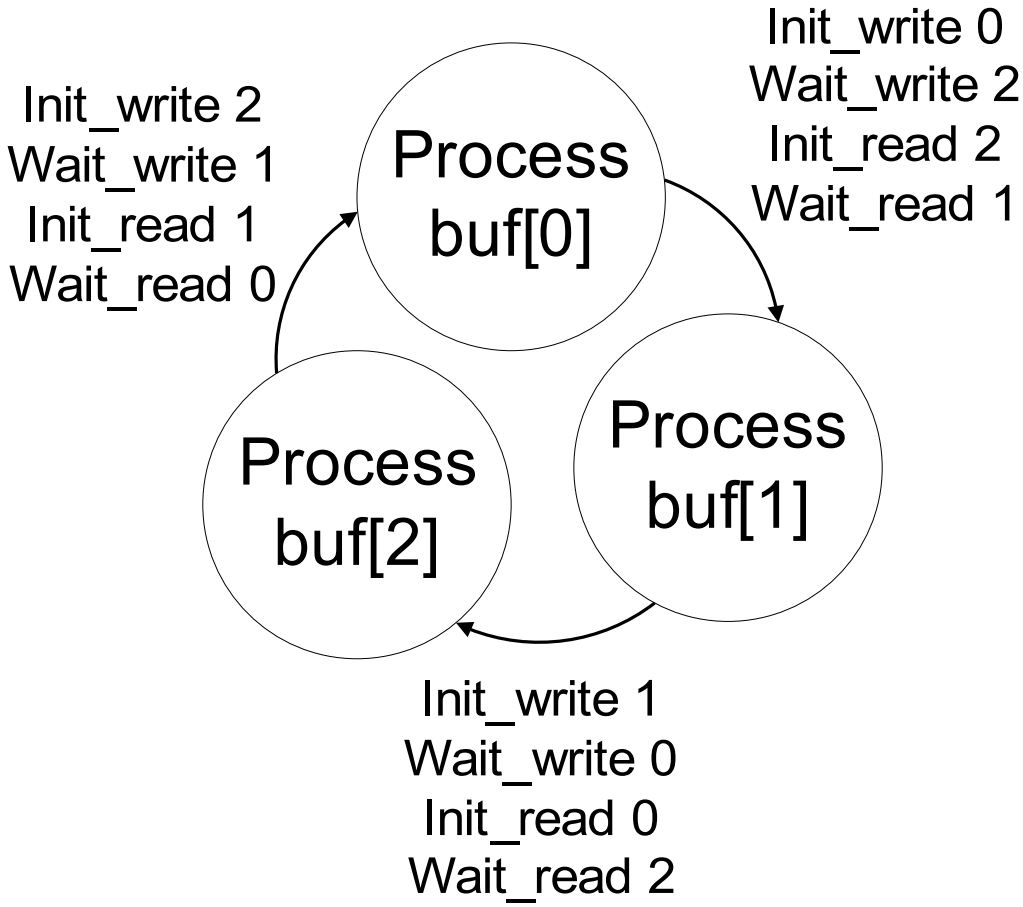
```
buf_indx = 0;
next_indx = 1;

bufs = init_get(buf_indx);

while(bufs >= 0){ // -1 - no data to process
   bufs = init_get(next_indx);
   wait_get(buf_indx);
   process(buf_indx);
   next_indx = buf_indx;
   buf_indx = buf_indx ^ 1;
}
```

Тройной буфер

```
i = 0;
end = 3;
while(end > 0){
   wait_write((i+1)%3);
   init_read((i+1)%3);
   init_write((i+2)%3);
   end -= wait_read(i);
   process(i);
   i++;
   i %= 3;
}
```



Память

- Пытайтесь держать данные в кэше
- Избегайте свопинга
  - Сжатие данных
  - Перевычисление данных
  - Избегайте дублирование данных

Размер данных

```
struct _t1_{
   char ch1;
   char ch2;
   int i;
}
```

```
struct _t2_{
   char ch1;
   int i;
   char ch2;
}
```

- Какой будет результат:
  - sizeof(_t1_)
  - sizeof(_t2_)

Размер данных

```
struct _t1_{
   char ch1;
   char ch2;
   int i1;
}
```

```
struct _t2_{
   char ch1;
   int i1;
   char ch2;
}
```

- sizeof(_t1_) = 8 bytes
- sizeof(_t2_) = 12 bytes
- sizeof(char) + sizeof(char) + sizeof(int) ==
  sizeof(char) + sizeof(int) + sizeof(char) ==
  6 bytes (!)
- Выравнивание данных на границу слова (4 bytes) для x86

Bytes	0	1	2	3	4	5	6	7	8	9	10	11
_t1_	ch1	ch2			i1							
_t2_	ch1				i1				ch2			

Упаковка данных

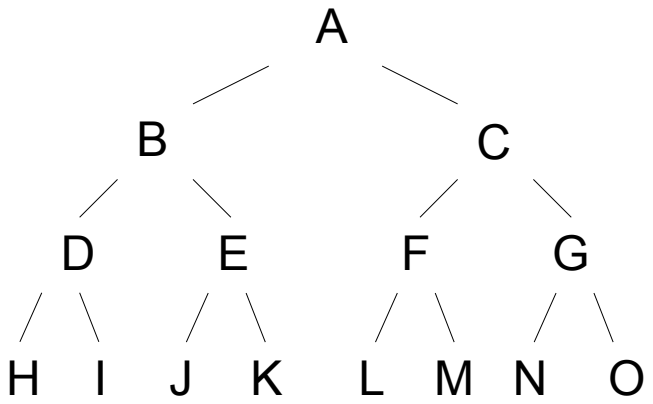
- Дан алфавит {'A', 'T', 'G', 'C', 'U'}
- Есть строки - РНК, ДНК. Задание - найти указанной подстроки в строках
- Решение проблем:
  - Читать все строки в символ *str [] и выполнить сравнение. "Программа работает слишком медленно" — жалоба студента.
  - "Гм! Давайте заменим символы цифрами и помести два символа в байт. ... используй асинхронное чтение при обработке ... " - мое предложение
  - Через некоторое время. "Это здорово. Асинхронное чтение не потребовалось. Упаковав данные я смог разместить все в памяти, и следовательно больше данных в кэше. Производительность достаточна для меня сейчас "

Больше данных в кэш

- При моделировании частиц кто-то может всю требую информацию поместить в структуру
  - ```
    struct data{
       int x,y,z;
       float mass;
       char color;
    } particles[number_particles];
    ```
- Цвет не требуется для вычислений!
  - Вынести в отдельный массив
  - Перейти от массива структур к структуре массивов
    ```

Деревья

- Дерево может быть представлено в виде массива
- Полезно дляr
  - Чтения дерева
  - Сохранения в файл
- Плохо для модфикации
- Подветки дерева:
  - Ind * 2 + 1
  - Ind * 2 + 2

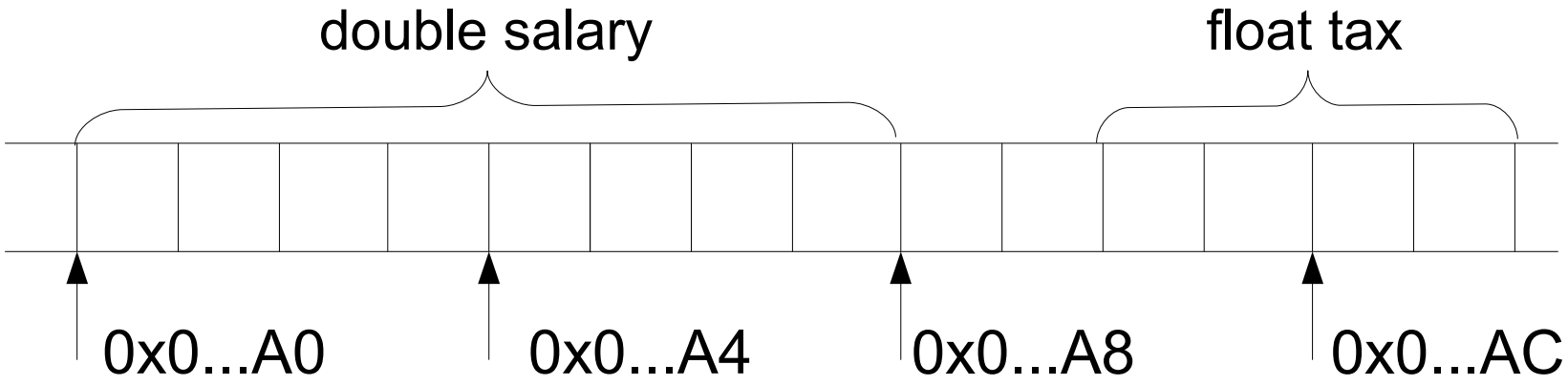


A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Выравнивание данных

Выравнивание данных - это то как данные организованы и доступны в памяти компьютера. Состоит два отдельных, но взаимосвязанных вопроса: выравнивание данных и заполнение структуры данных. В современных компьютерах, чтение\запись в адрес в памяти производится словами (например, 4 байт на 32-разрядной системе). Выравнивать данные - помещать данные со смещением в памяти равным нескольким словам, что увеличивает производительность системы. Для выравнивания данных может потребоваться вставка незначащих байтов между окончанием последней структуры данных и началом следующей

Выравнивание структур данных



- **Tax** не выравнена
- Для выборки **tax** CPU необходимо выбрать два слова и выполнить вычисление.
- Чтение\запись невыравненных данных может повлиять на производительность

Пример

float *a = (float*) malloc(128*sizeof(float));

- Не гарантируется, что **a** будет выравнен на границу слова
  - Используйте memalign(), valloc(), posix_memalign()
  - Выравнивание руками
- Предпочтительно выравнивание на границу строки кэша
- Для SSE инструкций данные должны быть выравнены на границу в 128bit

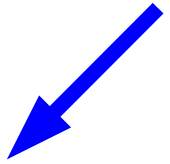
double mash[121][511];

- Что здесь не так?

Вызов функций

```
double Creconstruction::Integral(...){
    Cpoint3D KSI;
    // do something
    for(int i = 0; i<512; i++){
        // do something
        KSI = GetKSI(NewBasic, LAMBDA, ALPHA, 0, source);
        // do something
        if(Function(LAMBDA, KSI, circle, &cell)){
            // do something
        }
    }
    // do something
}

bool CReconstruction::Function(double LAMBDA, CPoint3D KSI,
                int circle, CMatrixElement *cell){
    // do something
}
```



Вызов функций

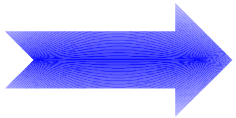
```
double
length(double x, double y){
   return sqrt(x*x+y*y);
}
          // do something
   for(i=0; i<N; i++){
      b[i] =
         length(a[i].x, a[i].y);
      // do something
   }
```

```
.globl length
      .type   length, @function
length:
.LFB2:
      pushq   %rbp
.LCFI0:
      movq    %rsp, %rbp
.LCFI1:
      subq    $48, %rsp
.LCFI2:
      movsd   %xmm0, -8(%rbp)
      movsd   %xmm1, -16(%rbp)
      movsd   -8(%rbp), %xmm0
...
      movsd   dst+820448(%rip), %xmm0
      movsd   dst+32784(%rip), %xmm2
      movapd  %xmm0, %xmm1
      movapd  %xmm2, %xmm0
      call    length
...
```

- Пометить функцию как inline
- Сделать функцию макросом

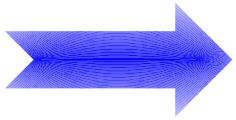
Рекурсия

```
unsigned Factorial(unsigned n){
   if(n == 0)
      return 1;
   return n*Factorial(n-1);
}
```



```
unsigned Factorial(unsigned n){
    unsigned res=1, i;
    for(i=1; i<=n; i++)
       res *= i;
    return res;
}
```

```
unsigned Fibonachi(unsigned n){
   if(n == 0 || n == 1)
      return 1;
   return Fibonachi(n-1)+
          Fibonachi(n-2);
}
```



```
unsigned Fibonachi(unsigned n){
    unsigned i;
    unsigned res[3];
    res[0] = res[1] = 1;
    for(i=2; i<=n; i++)
       res[i % 3] =
          res[(i-1) %3] +
          res[(i-2) %3];
    return res[n % 3];
}
```

Оптимизация под архитектура

- Ключи компиляции
  - gcc -mcpu={i586, i686,pentium, pentiumpro, k6, athlon, ...} -march -O{1,2,3,4}
  - icc {-tpp5, -tpp6, -tpp7, ...} -Ax{M,K,i,W} -O{1,2,3,4}
- Вставка ассемблерных инструкций
- Использование спец.библиотек (MKL)

- Возможно программа будет работать только на целевой платформе

Потоковые инструкции

- SIMD – Single Instruction Multiple Data
- Intel
  - SSE, SSE2, SSE3, SSE4
- AMD
  - MMX, 3DNow, SSE2, SSE3, SSE4
- SSE - 70 инструкций
- SSE2 добавлено 144 инструкций

Потоковые инструкции

```
void F(float *dest, float *src1, float *src2, unsigned len){
   unsigned i;
   for(i=0; i<len; i++)
      dest[i] = sqrt(src1[i]*src1[i] + src2[i]*src2[i]);
}

#include <xmmintrin.h>
void F_SSE(float *dest, float *src1, float *src2, unsigned len){
   unsigned i;
   __m128 m1, m2, m3, *arr1, *arr2, *arr3;
   arr1 = (__m128*)dest; // dest - 16 byte aligned
   arr2 = (__m128*)src1; // src1 - 16 byte aligned
   arr3 = (__m128*)src2; // src2 - 16 byte aligned
   for(i=0; i<len/4; i++){ // len divisible by 4
      m1 = _mm_mul_ps(*arr1, *arr1);
      m2 = _mm_mul_ps(*arr2, *arr2);
      m3 = _mm_add_ps(m1, m2);
      *arr3 = _mm_sqrt_ps(m3);
      arr1++; arr2++; arr3++;
   }
}
```

F_SSE() is 3-4 times faster then F()

Псевдо-оптимизация

- Делать что-то фоном, пока пользователь вводит текст (подгружать драйверы, проверять орфографию, делать бэкап, сканировать на вирусы и пр.)
- Помещать вычисления и GUI в разные потоки

Оптимизация параллельных программ

- Последовательная часть (см. выше)
- Коммуникации
  - Коммуникации должны занимать меньше времени чем вычисление
- Балансировка нагрузки
  - Не заставляйте один процесс ждать других
  - Переупорядочить начальные данные, шаги по времени, …
  - Master-slave

Инструменты

какая часть программы нуждается в оптимизации?

- Профилировщик - инструмент анализа производительности, который измеряет поведение программы, фиксирует как она выполняется
- История начинается с начала 1970-х годов
  - Использовались прерывания по времени для сохранения PSW и обнаружения "горячих точек" на IBM/360, IBM/370
- Выход:
  - Трассы
  - Выборки

Methods of data gathering

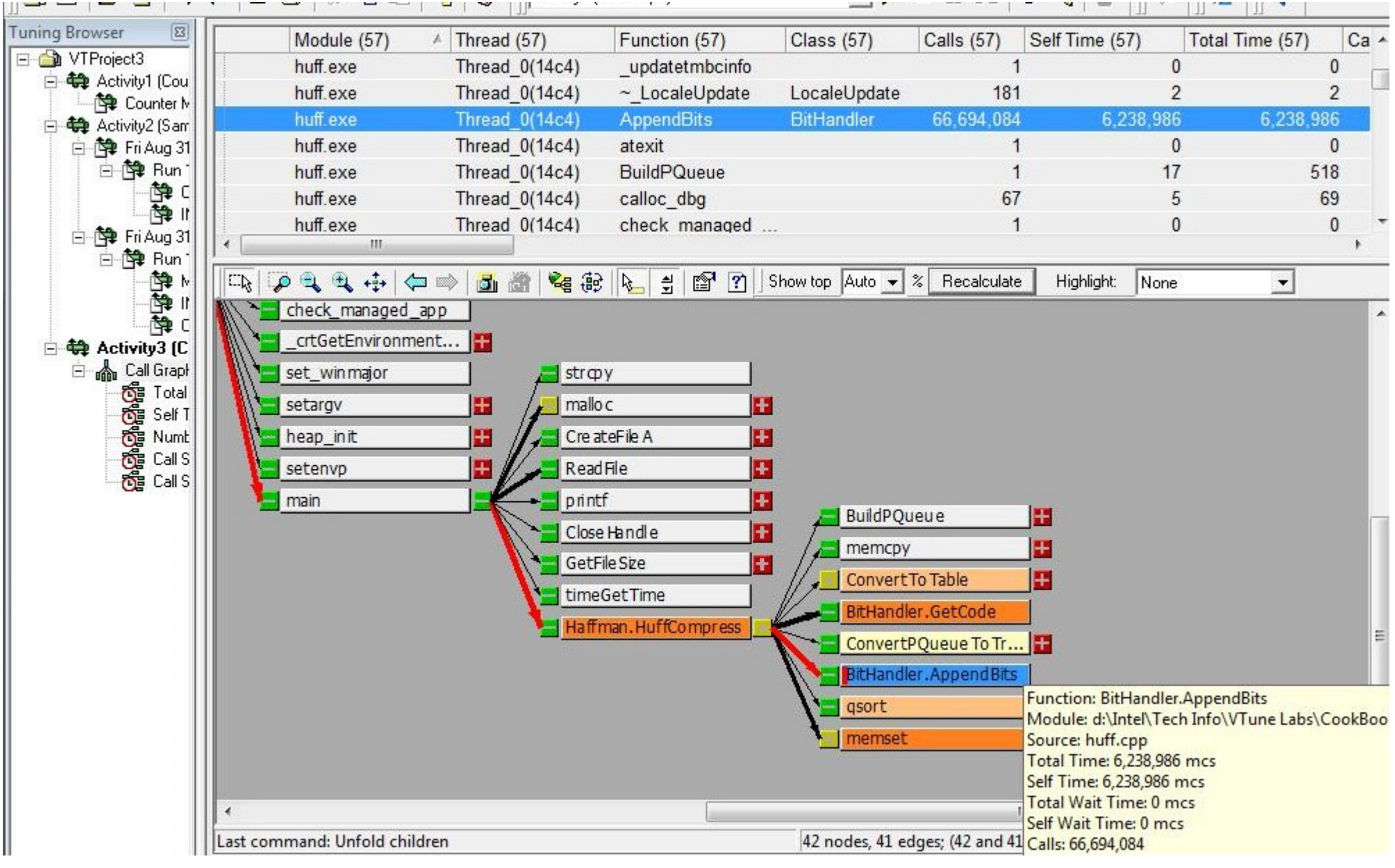
http://en.wikipedia.org/wiki/List_of_performance_analysis_tools

- Event based profilers
  - .NET, Java, Python, Ruby
- Statistical profilers
  - gprof
  - Oprofile
  - CodeAnalyst
  - VTune
  - Valgring
  - ...
- Hypervisor/Simulator
  - SIMMON, OLIVER

Gprof. Пример

```
Bash# gcc -Wall -O3 -g -pg mutation.c
Bash# ls gmon.out
gmon.out
Bash# gprof
%      cumulative    self                 self     total
 time  seconds      seconds      calls  s/call    s/call name
 43.41     121.94   121.94    6160896    0.00      0.00  MinusStr(...)
 20.27     178.89    56.94    3080448    0.00      0.00  MinusStr1(...)
 10.72     208.99    30.11        168    0.18      0.55  Read(...)
  8.45     232.73    23.74     299880    0.00      0.00  GetSost2(...)
  6.93     252.20    19.47     294000    0.00      0.00  BigMix0(...)
  5.66     268.09    15.89  246501360    0.00      0.00  IdeLet(...)
  2.65     275.53     7.44        168    0.04      1.11  BackMat(...)
  1.32     279.25     3.72  117600000    0.00      0.00  Mix(...)
  0.29     280.08     0.83         70    0.01      0.01  TransStr(...)
  0.23     280.72     0.64        168    0.00      1.67  EvalMah(...)
```

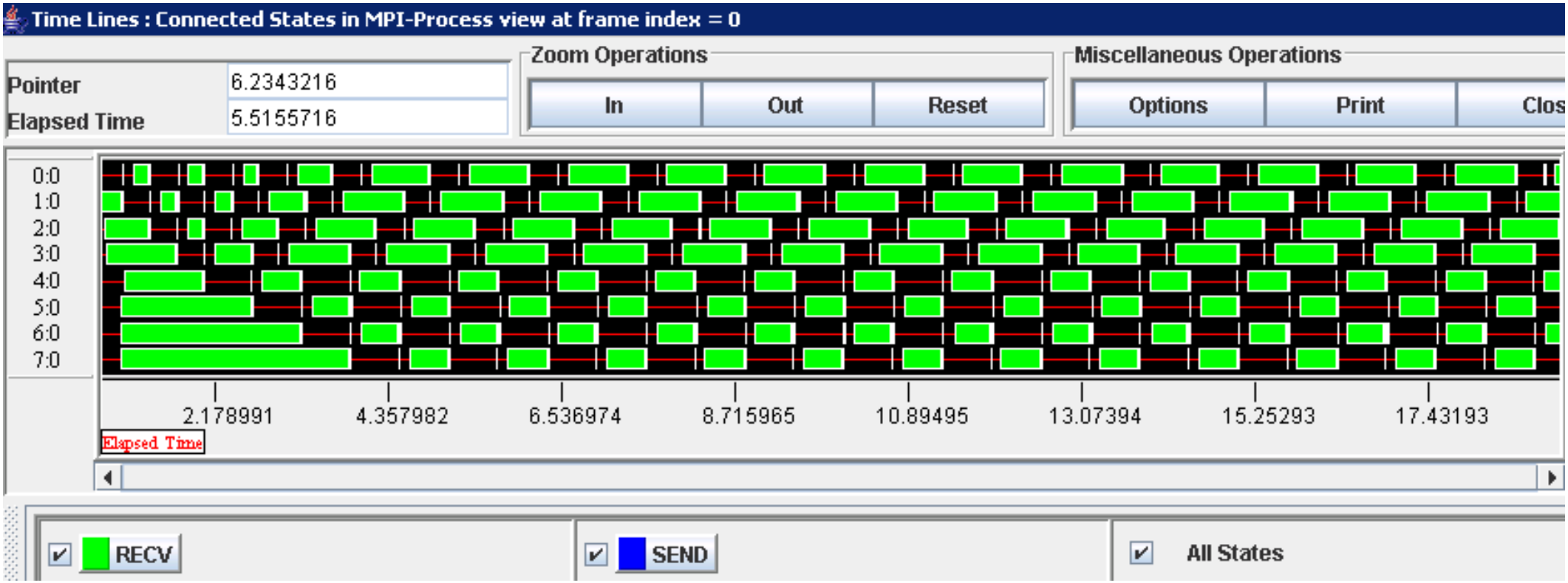
Intel VTune


Tuning Browser
VTProject3
Activity1 (Cou
Counter M
Activity2 (Sam
Fri Aug 31
Run
Fri Aug 31
Run
Activity3 (C
Call Graph
Total
Self T
Numb
Call S
Call S
Module (57)
Thread (57)
Function (57)
Class (57)
Calls (57)
Self Time (57)
Total Time (57)
huff.exe Thread_0(14c4) _updatetmbcinfo 1 0 0
huff.exe Thread_0(14c4) ~_LocaleUpdate LocaleUpdate 181 2 2
huff.exe Thread_0(14c4) AppendBits BitHandler 66,694,084 6,238,986 6,238,986
huff.exe Thread_0(14c4) atexit 1 0 0
huff.exe Thread_0(14c4) BuildPQueue 1 17 518
huff.exe Thread_0(14c4) calloc_dbg 67 5 69
huff.exe Thread_0(14c4) check_managed ... 1 0 0
Show top
Auto
Recalculate
Highlight:
None
check_managed_app
_crtGetEnvironment...
set_winmajor
setargv
heap_init
setenvp
main
strcpy
malloc
CreateFileA
ReadFile
printf
CloseHandle
GetFileSize
timeGetTime
Haffman.HuffCompress
BuildPQueue
memcpy
ConvertToTable
BitHandler.GetCode
ConvertPQueue To Tr...
BitHandler.AppendBits
qsort
memset
Function: BitHandler.AppendBits
Module: d:\Intel\Tech Info\VTune Labs\CookBoo
Source: huff.cpp
Total Time: 6,238,986 mcs
Self Time: 6,238,986 mcs
Total Wait Time: 0 mcs
Self Wait Time: 0 mcs
Calls: 66,694,084
Last command: Unfold children
42 nodes, 41 edges; (42 and 41

Инструменты для профилирования ПП

- HeNCE
- TRAPPER
- EDPEPPS
- GRADE
- AIMS (An Automated Instrumentation and Monitoring System)
- Vampir, VampirTrace
- Pablo Performance Analysis Toolkit Software
- Paradyn
- Jumpshot, Nupshot
- Puma
- CXperf

Jampshot


Time Lines : Connected States in MPI-Process view at frame index = 0
Pointer
6.2343216
Elapsed Time
5.5155716
Zoom Operations
In
Out
Reset
Miscellaneous Operations
Options
Print
Clos
0:0
1:0
2:0
3:0
4:0
5:0
6:0
7:0
2.178991
4.357982
6.536974
8.715965
10.89495
13.07394
15.25293
17.43193
Elapsed Time
RECV
SEND
All States