Program optimization

Alexey A. Romanenko
arom@ccfit.nsu.ru

What this course about?

In this course you will learn:

- Types of program optimization
- Approaches for program optimization
- Profiling tools

Optimization

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

On ANY architecture!

Types of optimization

- Performance
- Memory
- Scalability
- Communications
- etc.

- We address mainly to performance optimization

Statements

- Donald Knuth
  - Premature optimization is the root of all evil
- If your code already works, optimizing it is a sure way to introduce new, and possibly subtle, bugs
- Optimization tends to make code harder to understand and maintain
- Some of the optimization techniques increase speed by reducing the extensibility of the code
- A lot of time can be spent optimizing, with little gain in performance, and can result in obfuscated code
- If you're overly obsessed with optimizing code, people will call you a nerd behind your back

Statements

- Faster program, more memory required.
- Making more faster program required more time for optimization.
- Optimizing code for one platform may actually make it worse on another platform

BUT!!!!!

- Michael Abrash (Quake, Unreal Tournament, Space Strike, Half-life, etc.):
  - Performance must always be measured
  - Any optimization, user can feel, should be done.

Does compilers make an optimization?

- They does. But
  - Compiler do not have database for algorithms
  - Compiler know nothing about field of investigation
  - It's impossible for compiler to look through your program totally
  - Worth implementation couldn't be fix by any compiler
  - What if your program should executed as you wrote it?

Stages of optimization

- Program design
  - Selecting algorithms and data structures
  - Speed up factor – 100 ...1000+
- Program implementation
  - Programming language
  - Speed up factor – 10 ...100+
- Program profiling
  - Turn program for the architecture
  - Speed up factor – 1 ...10+ (not for CELL)

Example

```
s, i, N - integer

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

If N equal 40.000.000.000, it takes

- ~ 6 hours on Intel PIII-933!
- ~ 1 hour on Xeon Dualcore 2.4GHz (OpenMP, Pthreads)

- Let's take Quadro-core CPU and 4 CPU system board.

Solution

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] – index of right edge
//[N/(k+1)] – index of left edge

i = N;
while(i>0){
    //m – number of elements with the same value
    m = i - N/(N/i + 1);
    s +=(N/i) * m;
    i -= m;
}
```

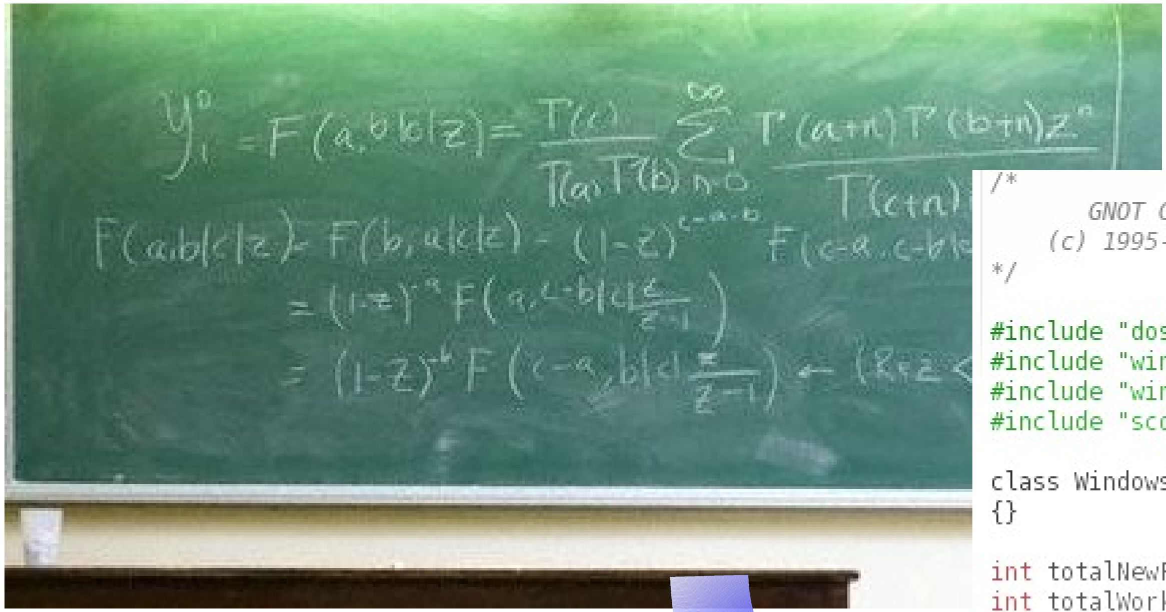
Less then 1 sec!

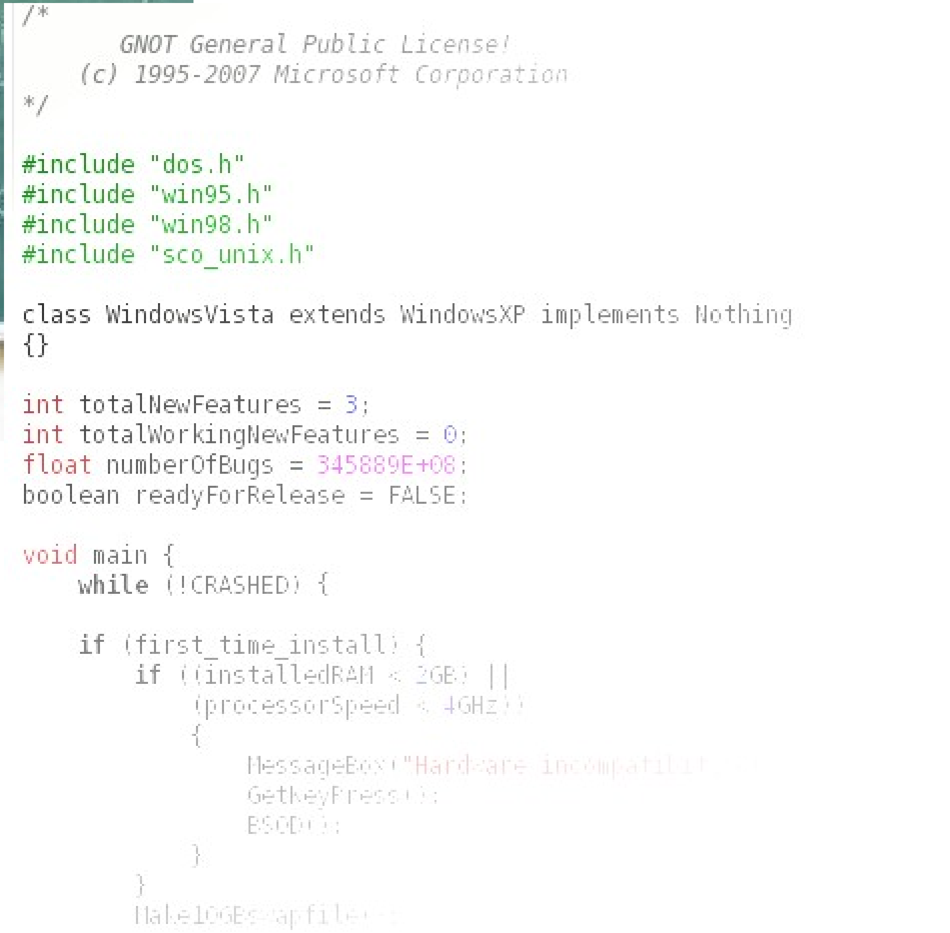
Selecting algorithm

- Read books
  - 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.
  - Field related works.
- Find symmetry
- Think of border conditions
- Apply your imagination, thing out of box.

If your program need to be run once, perhaps, you don't need to spend your working hours for its optimization. It would be better to spend this time with you friends while program work.

Program implementation





```
/*
     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 incompatib
                GetKeyPress();
                BSOD();
            }
        }
        Make10GBswapfile();
```

Program implementation

- Loops
- Memory I/O
- Function calls
- Passing parameters
- etc.

Loops

- Rule of “10-90” - 90% of time you program spends in 10% of the code.
- As a rule, this 10% of code is loops.
- Optimizing loops you can significantly speed up your program

Loops optimization

- Move conditions outside of the loop
- Eliminate data recalculations
- Keep all data in cache
- Place simples operations inside the loop
  - Unroll loops

Conditions inside the loop

```
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
```

Conditions inside the loop

- Unnecessary operations inside the loop
  - Comparison
  - Jump
- Stall processor's pipeline on wrong prediction

Data recalculation

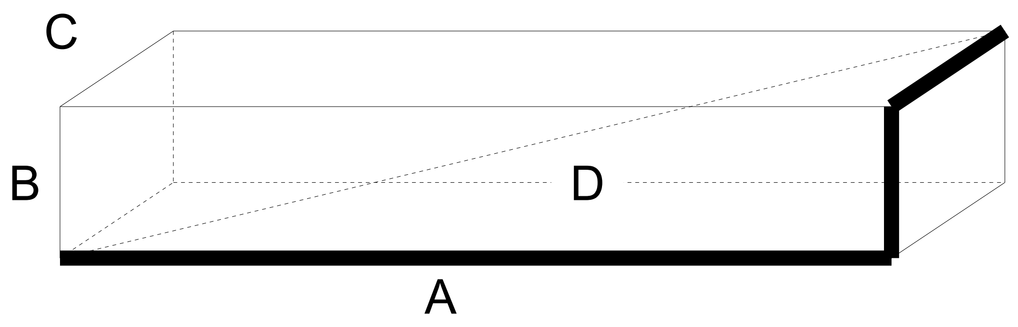
```
for(i=0; i<N; i++){
   s += F(i)*dx;
}
```

$\Rightarrow$

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

Problem

- Parallelepiped is given
- Edges of parallelepiped are A, B, C – integer
- 1 <= A, B, C, <=250
- Find all parallelepipeds, where main diagonal D is integer, and D<=250



Solution 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 ==  A*A + B*B + C*C){
               my_print(A, B, C, D);
            }
         }
      }
   }
}
```

Notes

- A*A could be calculated in advance;
- If one parallelepiped was found (ABC), we know other five: ACB, BAC, BCA, CAB, CBA. Modify my_print function;
- We can check edges which meet the following equation: A<=B<=C

Solution 2

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

Twice times faster!

Keeping data in cache

```
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
```

Array elements

- 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), ...

- Data in cache are stored in cache lines
- Cache line is from 8 to 512 bytes
- Cache line contains several stored elements
- Fetching data from cache faster then fetching data from memory

Keeping data in cache

```
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
```

Simplest operation

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

$\Rightarrow$

```
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];
```

$\Rightarrow$

```
for(i=0; i<K; 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];
}
```

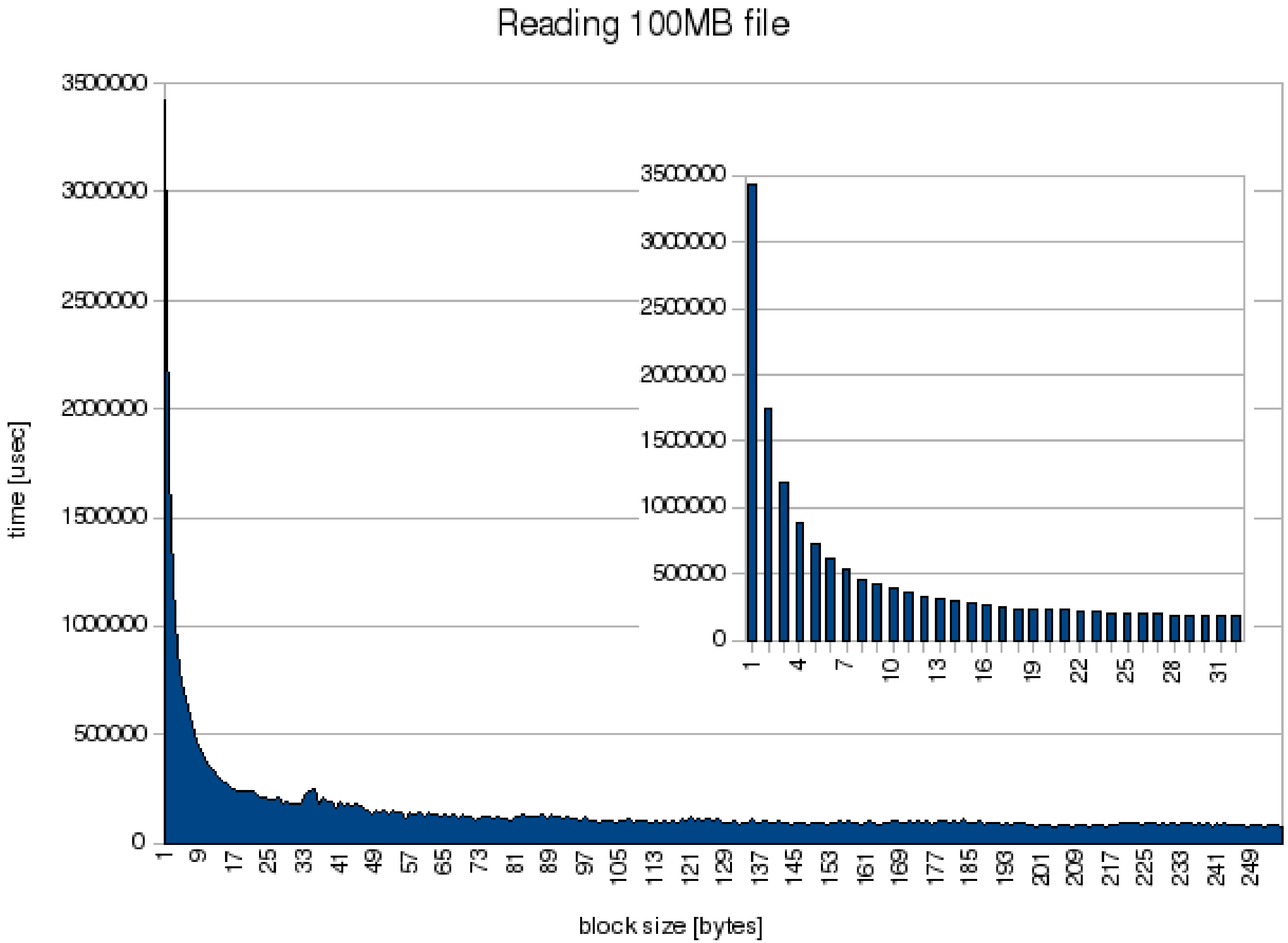
File I/O operations

Reading data

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

⚠ Do not read files by chars

Reading data diagram



Reading data

```
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++];
}
```

- In Unix-like systems use mapping of file onto memory
  - mmap, munmap

Writing data

- Use buffered output
- Do not write a lot to console

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

Read and write asynchronously

- 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 *);

Double buffer

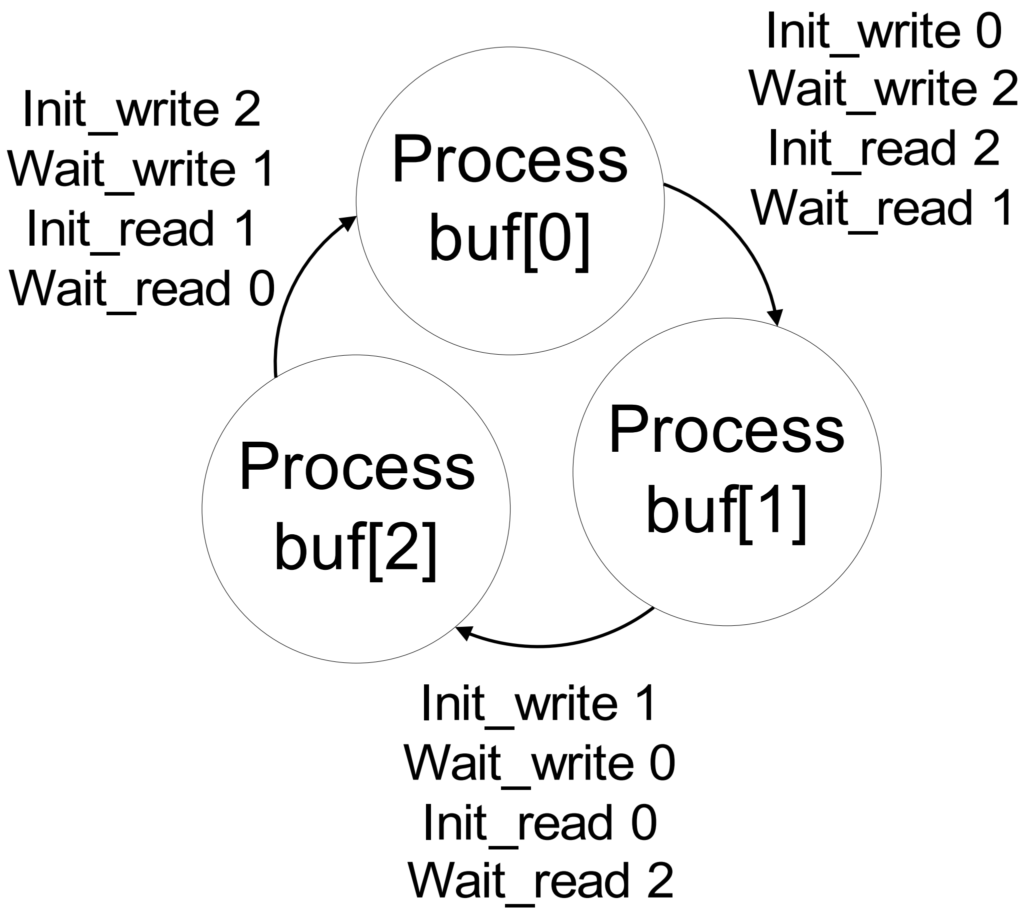
```
buf_indx = 0;
next_indx = 1;

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;
}
```

Triple buffer

```
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;
}
```



Memory

- Try to place all you data into CPUs cache
- Try to keep all you data in RAM
- Avoid swapping
  - Compress data
  - Data recalculation
  - Illuminate data duplication

Size of Data

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

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

- What is the result of:
  - sizeof(_t1_)
  - sizeof(_t2_)

Size of Data

```
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 (!)
- Data alignment – word (4 bytes) by default for x86

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

Packing data

- ABC is given {'A', 'T', 'G', 'C', 'U'}
- There are strings – RNA, DNA. Task is to find specified substring in the strings
- Problem solving:
  - Read all strings into **char *str[]** and perform comparison. “Program works too slowly” - students complain.
  - “Hmm! Let's change chars with numbers and place two chars into a byte. ... use asynchronous read while processing...” - my suggestion
  - After a while. “That's great. No asynchronouse read required. By packing data I could allocate all of them in memory and therefor more data in cache. Performance is enough for me now”

Keep more data in cache

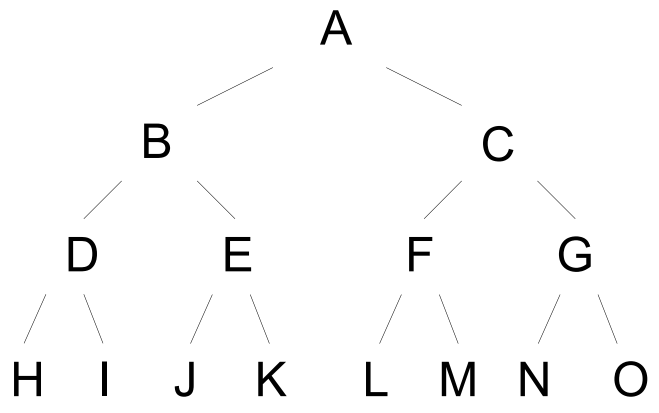
- For modeling particles one could place information about each particle into the following structure:

  -
    ```
    struct data{
        int x,y,z;
        float mass;
        char color;
    } particles[number_particles];
    ```
- Color is not required for calculation
  - Move color into separate array
  - Turn from array of structs to structs of array

Lists

Trees

- Tree could be represented as array
- Useful for
  - reading data in tree
  - Storing tree into file
- Not good for tree modification
- Sub-trees roots are:
  - Ind * 2 + 1
  - Ind * 2 + 2

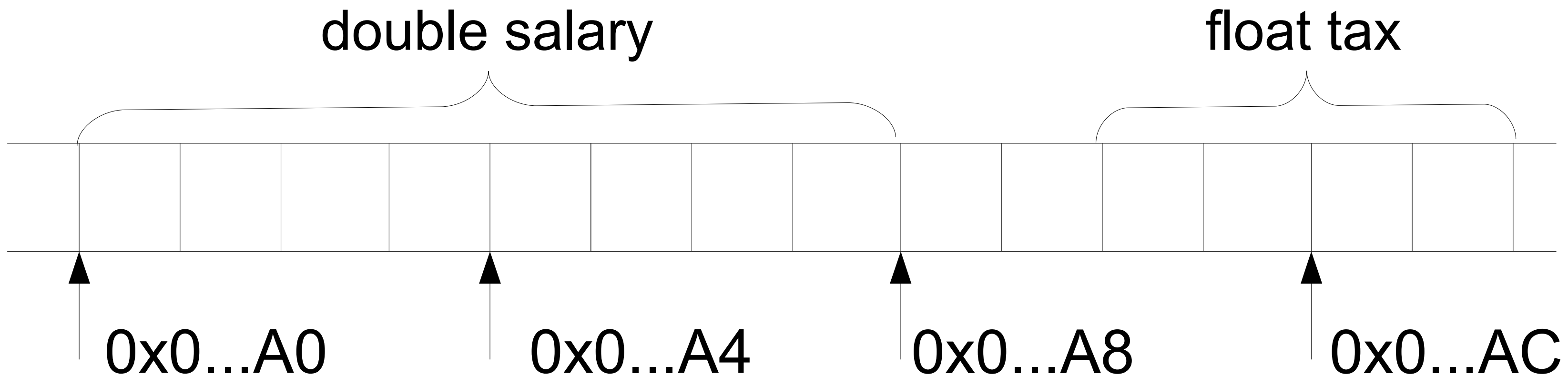


A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Data structure alignment

Data structure alignment is the way data is arranged and accessed in computer memory. It consists of two separate but related issues: **data alignment** and **data structure padding**. When modern computers read from or write to a memory address, it will do this in word sized chunks (e.g. 4 byte chunks on a 32-bit system). Data alignment is to put the data at a memory offset equal to some multiple of the word size, which increases the system's performance due to how the CPU handles memory. To align the data, it may be necessary to insert some meaningless bytes between the end of the last data structure and the start of the next, which is data structure padding.

Data structure alignment



- **tax** is not aligned
- To fetch **tax** CPU need to fetch two words and do some calculations.
- Reading and writing unaligned data cause some performance problems

Examples

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

- There is no guaranty **a** is word aligned
  - Use memalign(), valloc(), posix_memalign()
  - Align address by hands
- Alignment to cache line border is preferred
- For SSE routines data should be 128bit aligned

```
double mash[121][511];
```

- What's wrong here?

Functions call

```
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
}
```

Functions call

```
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
...
```

- Mark length function as inline
- Make length function as define

Recursion

```
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];

}
```

Architecture specific optimization

- Compilator keys
  - 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}
- Rewrite routines on ASM, use specific CPU instruction
- Use specific optimized libraries (MKL)

⚠

- Optimized program would work on target platform only

Stream instructions

- SIMD – Single Instruction Multiple Data
- Intel
  - SSE, SSE2, SSE3, SSE4
- AMD
  - MMX, 3DNow, SSE2, SSE3, SSE4
- SSE has 70 instruction
- SSE2 adds 144 instructions

Streaming instructions

```
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()

Pseudo optimization

- Do something while user type a text (load drivers, spell checking, saving backup, scan for viruses)
- Move calculations and visualization into different threads

Parallel program optimization

- Sequential parts (see above)
- Communications
  - Communication should take less time than calculations
- Load balancing.
  - Don't make one process to wait other one
  - Rearrange initial data, rearrange each time step, ...

Tools

Which part of your program should be optimized?

- A **profiler** is a performance analysis tool that measures the behavior of a program as it executes, particularly the frequency and duration of function calls
- Profilers history starts from early 1970s
  - Used time based interrupts for saving PSW and discover “hot spots” on IBM/360, IBM/370
- Output:
  - Trace
  - Sampling

Methods of data gathering

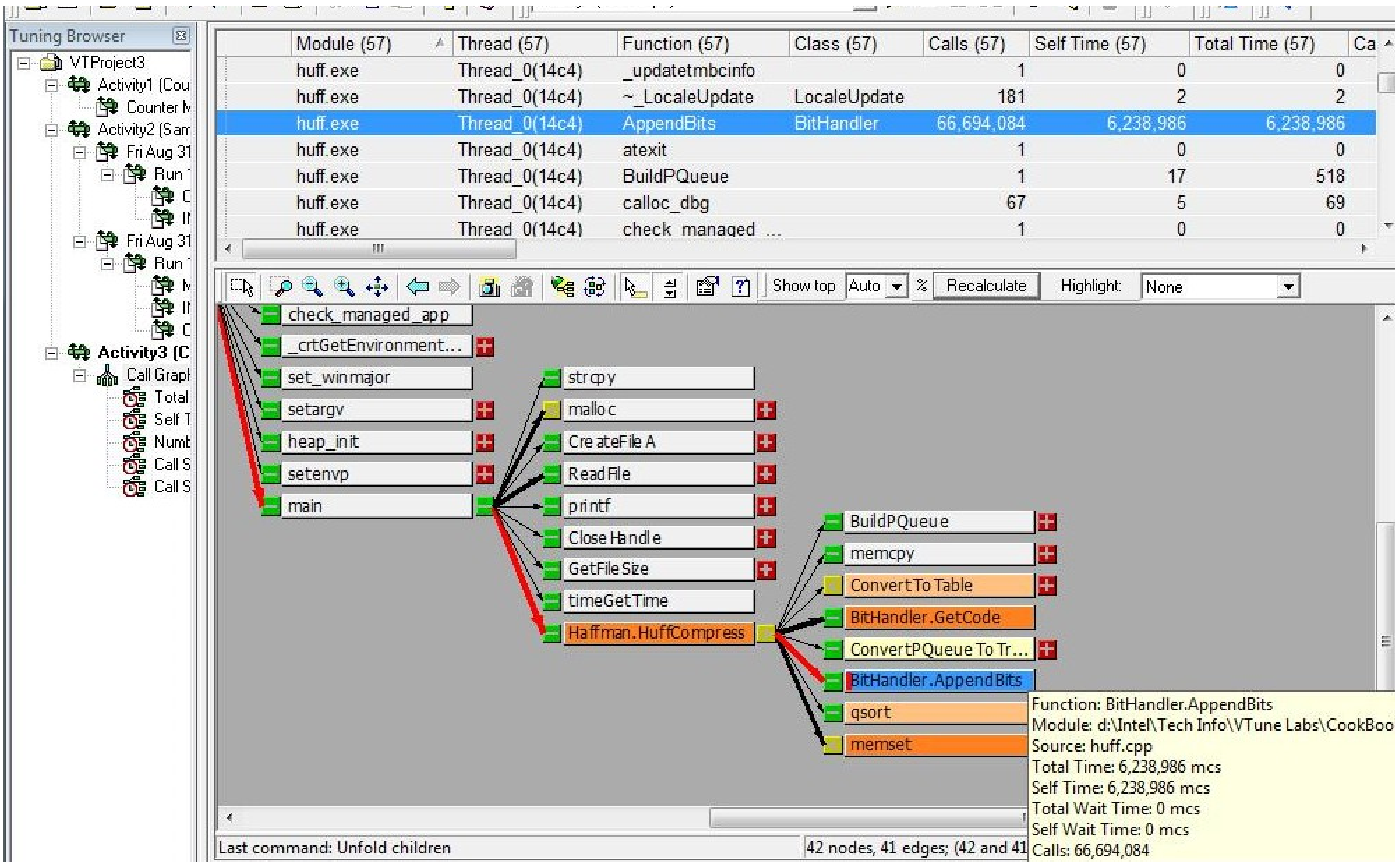
http://en.wikipedia.org/wiki/List_of_performance_analysis_tools

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

Gprof example

```
Bash# gcc -Wall -O3 -g -pg mutation.c
Bash# ./a.out
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



Module (57)	Thread (57)	Function (57)	Class (57)	Calls (57)	Self Time (57)	Total Time (57)	Ca
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	

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

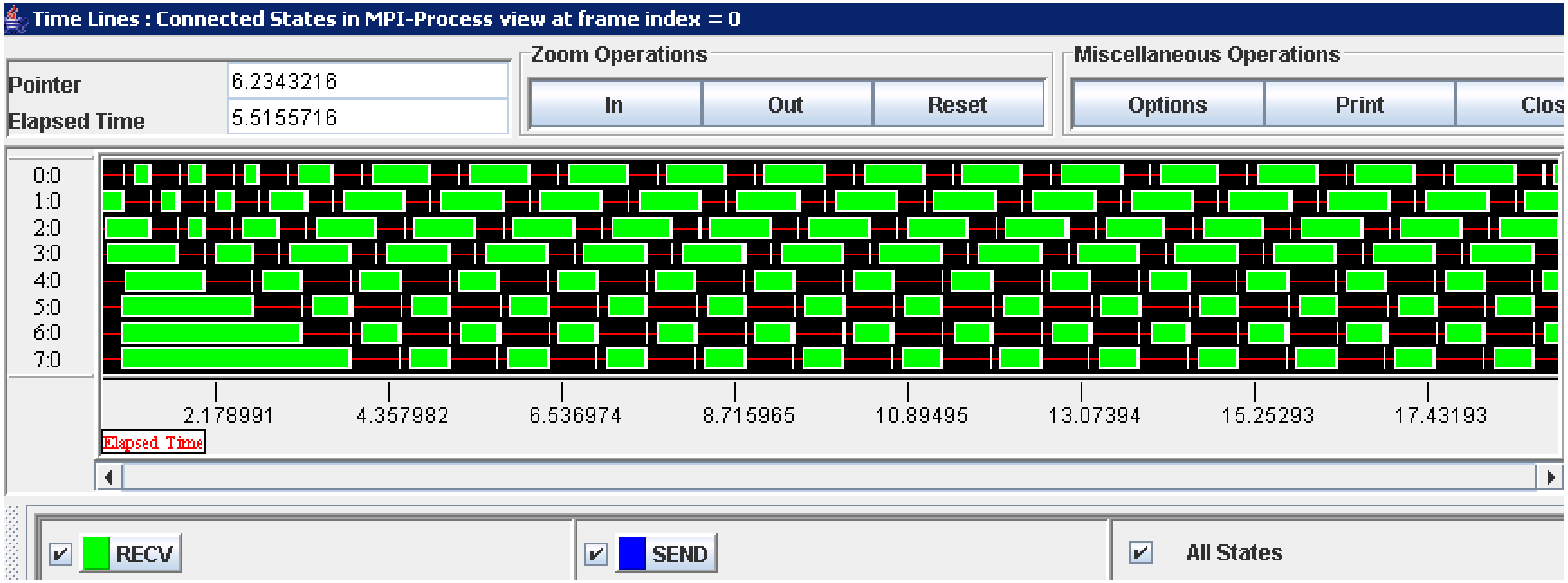
Last command: Unfold children

42 nodes, 41 edges; (42 and 41

Parallel program debugging and optimization tools

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

Jampshot



Pointer 6.2343216
Elapsed Time 5.5155716

✓ 🟩 RECV
✓ 🟦 SEND