

<http://u.arizona.edu/~dpsaltis/Phys305/text.html>

Developing the C Programming Language

The developers of the C programming language, Ken Thompson (sitting) and Dennis Ritchie, in front of a PDP-11/20 computer at the Bell Labs, in 1972 (Scientific American, March 1999). The PDP-11/20 computer could manage 64 Kbytes of memory at any given time



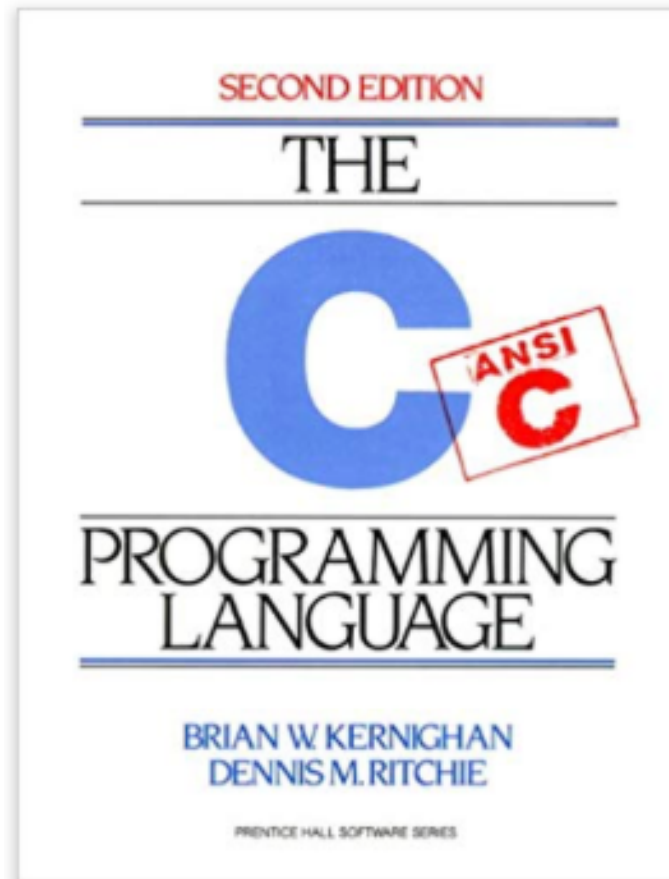
The International Obfuscated C Code Contest

The C language offers an unprecedented flexibility both in the construction of a computer program and in the presentation of its source code. Since 1984, the International Obfuscate C Code Contest^[5] has been rewarding the most “obscure/obfuscated C program, which shows the importance of programming style, in an ironic way”. The program below, which was one of the winning entries in 1995, asks the user for an integer and calculates its factorial (you should try it!). Albeit legitimate, this is clearly not the way to write a computer program that is easy to understand or that allows for easily spotting potential mistakes.

```
#include <stdio.h>

#define l111 0xFFFF
#define l11 for
#define l1111 if
#define l111 unsigned
#define l111 struct
#define l1111 short
#define l1111 long
#define l1111 putchar
#define l1111(l) l=malloc(sizeof(l1111 l1111));l->l1111=1-1;l->l1111=1-1;
#define l1111 *l1111++=l111%10000;l111/=10000;
#define l1111 l1111({l1->l1111}){l1111(l1->l1111);l1->l1111->l1111=11;}\
l1111=(l1=l1->l1111)->l11;l1=1-1;
#define l111 1000

l1111,*l1111 ;l111
l111});;main (l111 l1111
l1,*l111,* malloc ( ) ; l111
l1111 l11,l1 ,l;l11 l1111 *l111,*
=1-1 ;l< 14; l1111("\t\"8)>l\"9!.)>v1"
);scanf("%d",&l);l1111(l11) l1111(l111
l11(l11->l11[l1-1] =1)=l111;l11(l11
++l11){l1=l111; l111 = (l111=(
l1;l1111 =( l11=l1)->l11;l
);l11;l111-> l1111||l111!-
+=l11*l111++ ;l1111 l1111
l1111 l1111={ l111 =l111->
};l11;l111; )l1111 l1111
{ l1111 } * l1111=l111;}
l11(l=(l1=1- l);(l<l111)&&
(l1->l11[ l] !=l111);++l);
l1->l1111,l= l111){l11(--l
++l)printf( (l1)?(l1%19
19,"\n%04d") );:"%4d",l1->
l111 l1111 {
l111 l1111 *
l1111 l11 {
*l111,*l11,*
l1111 l111 ;
l1111;l11(l
l111)^'L',++l
)(l1=l11)->
=1+1;l11<=1;
l111=l11)->
l1=(l111=1-1
*l111;){l111
(++l1>l111){
l1111)->l11;
(++l1>=l111)
l11 ( ;l1;l1=
;l1>=1-1;--l,
?"%04d":(l1=
l11[l] ) ; }
l1111(10); }
```



```
#include <stdio.h>           // incorporates libraries
                             // for input/output
int main(void)              // begin main program
{
    printf("Hello World!\n"); // print on screen Hello World!

    return 0;                // normal end of program
}
```

```
gcc hello.c -o hello.c
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

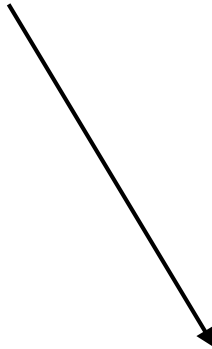
```
    float x;
```

```
    for(x=0.0;x<=1.0;x+=0.1)
```

```
        printf("x=%f f(x)=%f\n",x,x*x);
```

```
    return 0;
```

```
}
```



```
printf("x=%10.8f f(x)=%10.8f\n",x,x*x);
```

Rounding errors may have deadly consequences

On February 25, 1991, during Operation Desert Storm, a Scud missile fired by the Iraqi army hit a U.S. army barracks in the Dhahran Air Base in Saudi Arabia, killing 28 soldiers. The air base was protected by a Patriot Air Defense System, which nevertheless failed to track and intercept the incoming Scud missile. An investigation of the incident by the U.S. army revealed that accumulation of rounding errors in the Patriot tracking software was responsible for the failure of the interception^[5].

Each battery of the Patriot Air Defense system consists of a ground based radar and eight missile launchers. The radar detects airborne objects and tracks their motion. In order to distinguish between different types of objects and, in particular, in order to identify ballistic missiles, a weapons control computer calculates the expected trajectory of the missile. If the airborne object detected by the radar follows the expected trajectory, a Patriot missile is fired to intercept it.

The weapons control computer uses an internal clock to keep track of time and perform the calculations of the trajectories of the detected objects. The time is stored in 24-bit variables (registers) in increments of 0.1 seconds. In the binary system, however, $(0.1)_{10} = (0.0001100011\dots)_2$ and hence any calculation involving this time increment needs to be rounded. On February 25, 1991, after approximately 300 hours of continuous operation, the control computer had accumulated enough rounding error that it did not predict accurately the trajectory of the Scud missile. As a result, the missile was not identified and a Patriot was not fired to intercept it.



Data type conversion errors can be very costly

Ariane 5 is the primary launch vehicle of the European Space Agency (ESA). that operates from the Guiana Space Center near Kourou in the French Guiana. Its first successful operational flight took place in December 1999, when it carried to space the European X-ray Multi Mirror (XMM) satellite. Its first test launch, however, on June 4, 1996 resulted in failure, with the rocket exploding 40 seconds into the flight.

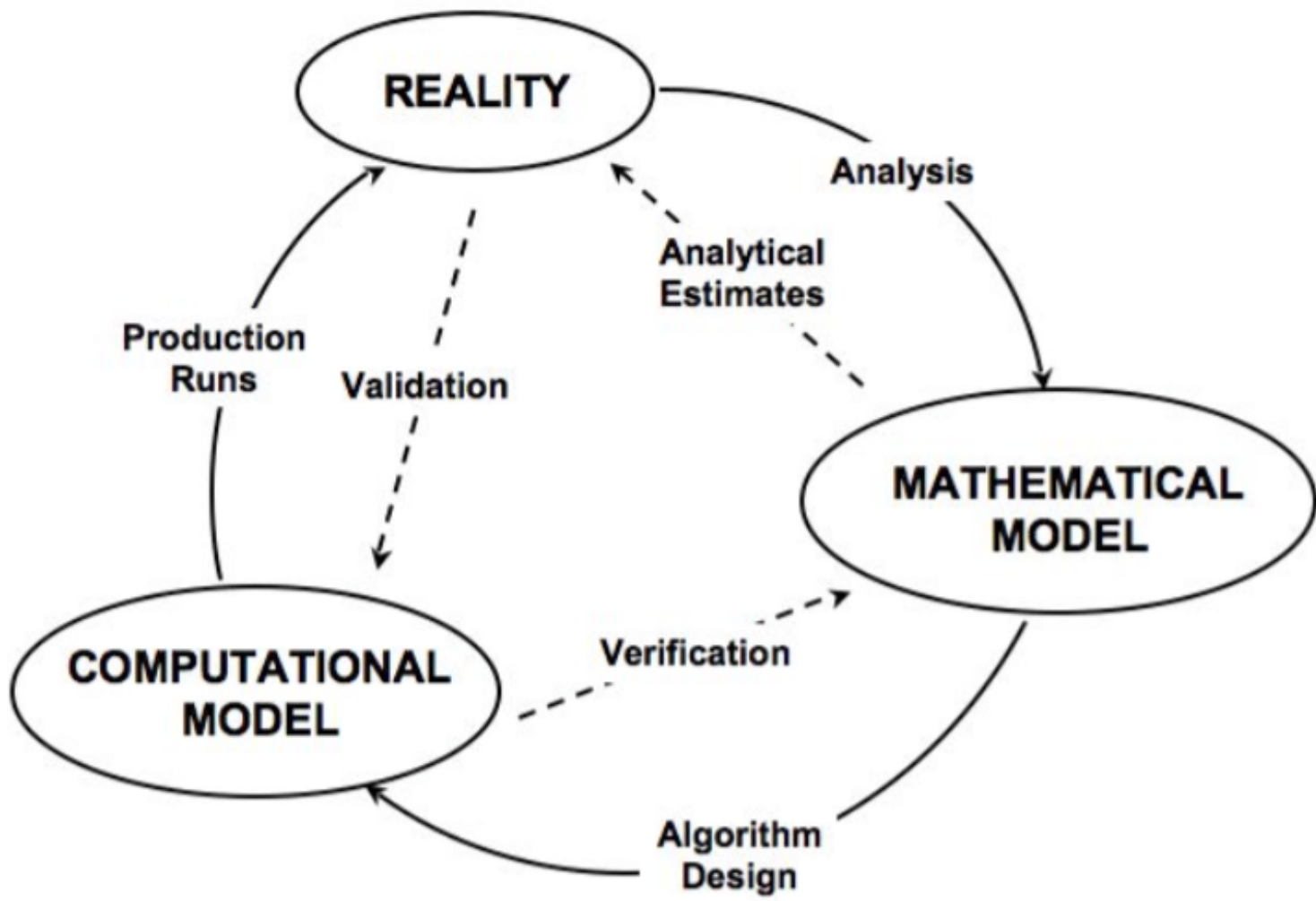
A study of the accident by an inquiry board^[4] found that the failure was caused by a software problem in the Inertial Reference System that was guiding the rocket. In particular, the computer program, which was written in the Ada programming language and was inherited from the previous launch vehicle Ariane 4, required at some point in the calculation a conversion of a 64-bit floating point number to a 16-bit integer. The initial trajectory of the Ariane 5 launch vehicle, however, is significantly different than that of Ariane 4 and the 16 bits are not enough to store some of the required information. The result was an error in the calculation, which the inertial system misinterpreted and caused the rocket to veer off its flight path and explode. The cost of the failed launch was upwards of 100 million dollars!

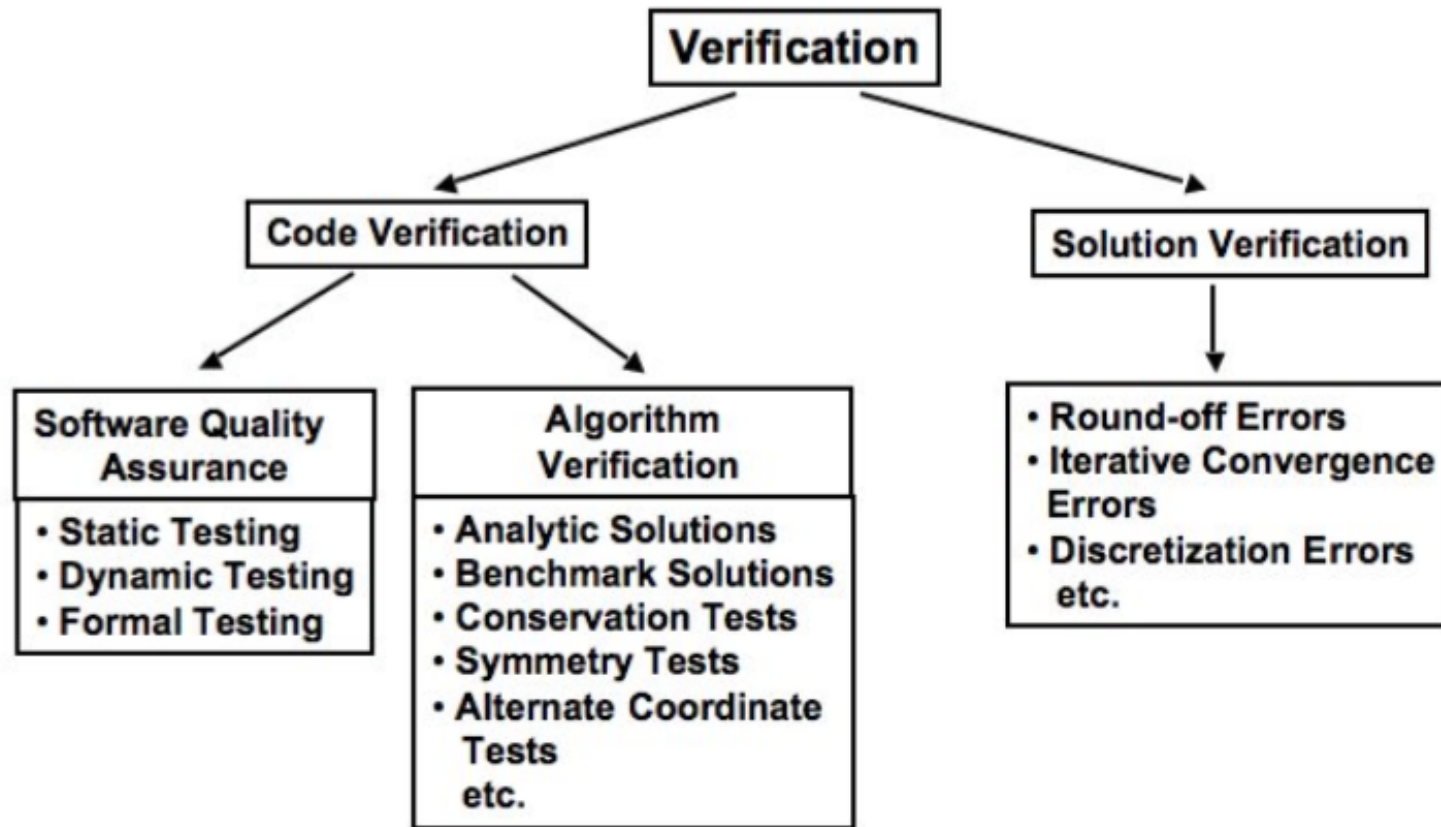


The First Real Computer Bug

In modern computer jargon, a bug is a mistake introduced inadvertently in an algorithm that prevents it from completing its execution or from providing accurate results. The process of correcting an algorithm from such mistakes is called *debugging*. Although this term is used figuratively today, the operation of early computers was often affected by insects flying between their vacuum tubes! The first “real” computer bug was found in 1947, in the computer Mark II at Harvard University. Computer engineers taped the bug in their log-book, which can be found today at the Smithsonian Museum of American History.



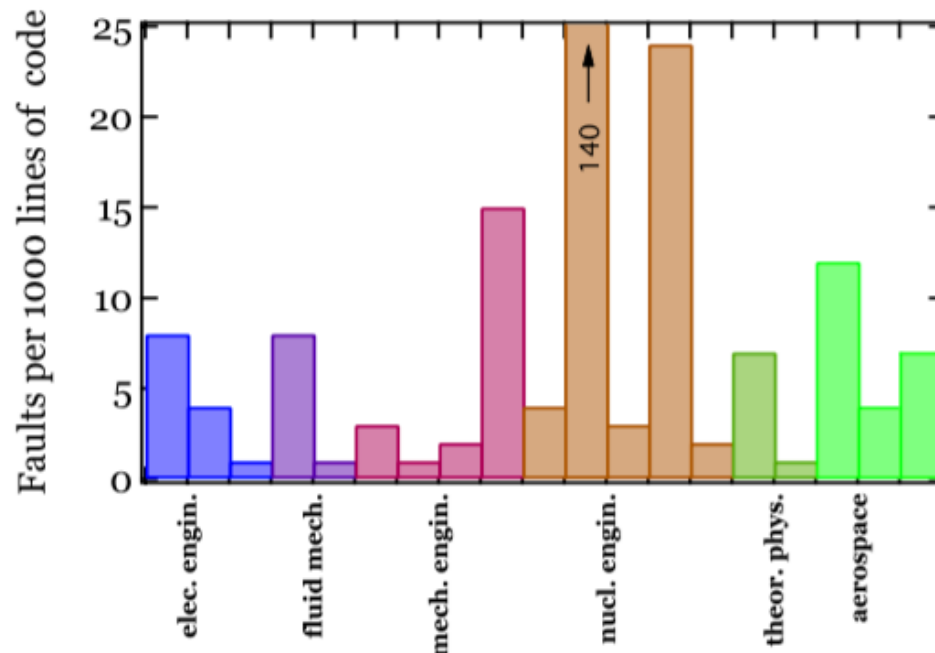




No code is perfect; some are not nearly correct!

In 1997, an experiment was conducted that aimed to investigate the number of serious faults found in mature computer codes used in a very large number of disciplines, from medical imaging to aerospace^[4]. More than 100 computer packages were analyzed with a combined total of more than 5 million lines of computer code in C and in Fortran. All of these packages were considered to have been fully tested by their developers.

A static analysis was performed on each package and every identified fault was given an appropriate weight, depending on its severity. The following figure, which was adopted from the study, shows that 1 to 25 serious faults per 1000 lines of code are commonplace in computational physics and engineering. One particularly disturbing result involves a package in nuclear engineering, which reached 140 faults per 1000 lines of code. As the author of the study remarked, this code “in spite of the aspirations of its designers, amounted to no more than a very expensive random number generator”.



```
#include<stdio.h>
#include<math.h>
#include<time.h>

#define Nrep 1000000

int main(void)
{
    double x=1.3,a;
    double time,Mflops;
    int i;
    clock_t ticks1, ticks2;

    ticks1=clock();

    for (i=1;i<=Nrep;i+=2)
    {
        a=x+x;
    }
    ticks2=clock();

    time=(1.0*(ticks2-ticks1))/CLOCKS_PER_SEC/Nrep;
    Mflops=1.e-6/time;

    printf("it took %e seconds\n",time);
    printf("this corresponds to %f MFLOPS\n",Mflops);

    return 0;
```

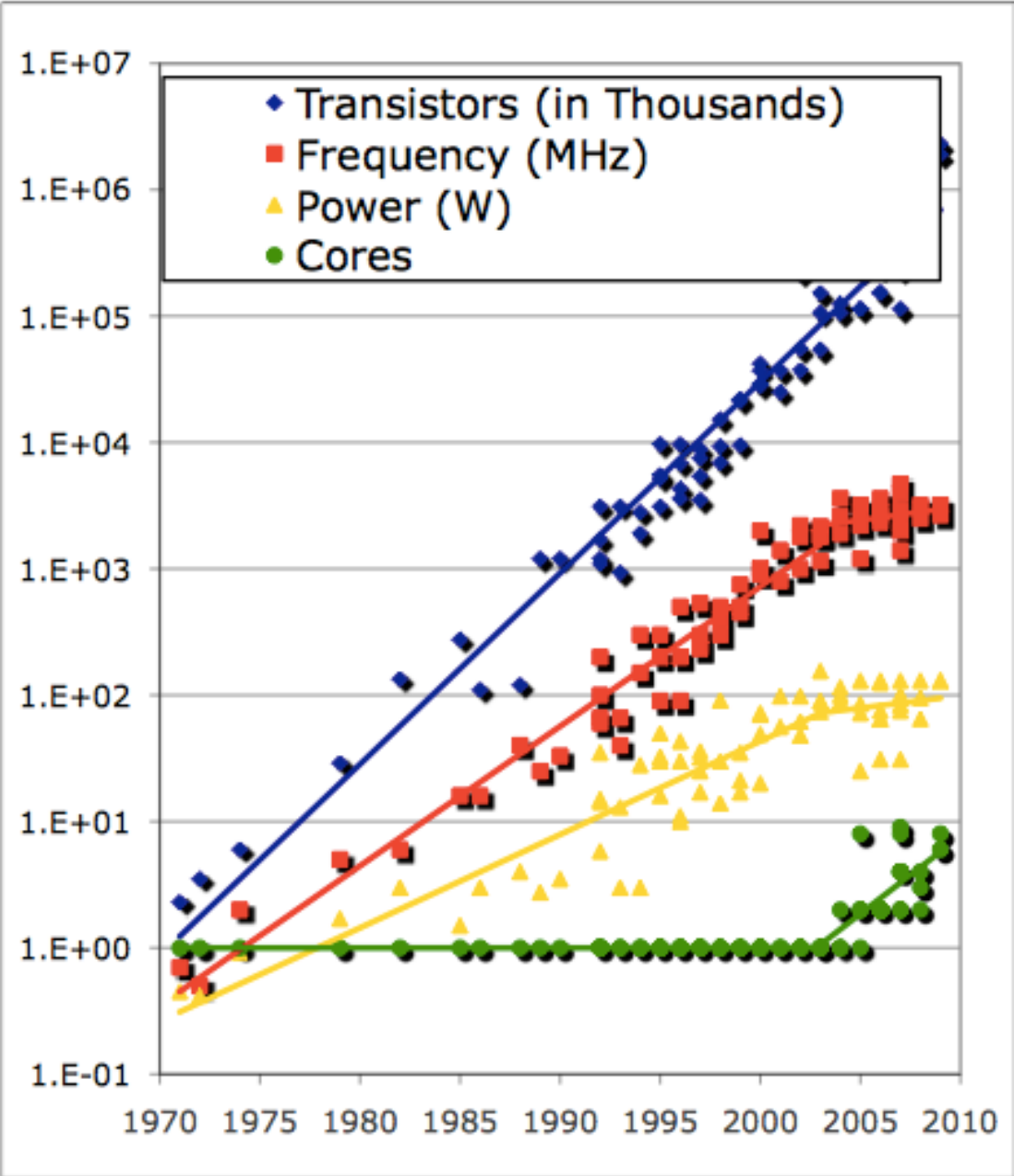
```
a=i*x;  
a=i/x;  
a=i/x/x;  
a=i/(x*x);
```

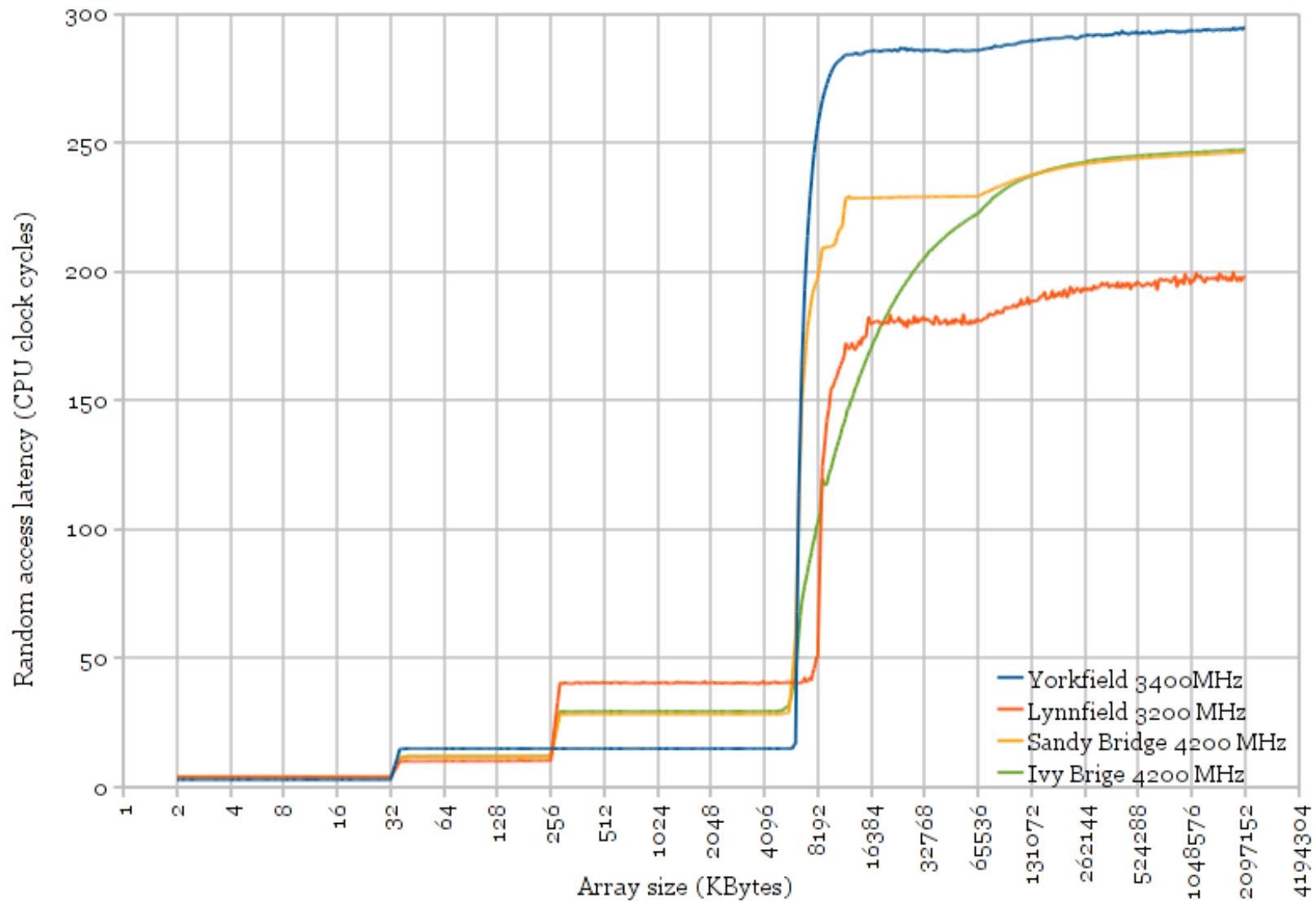
```
a=sin(x)*sin(x)+2.*cos(x)*cos(x);  
a=1.+cos(x)*cos(x);
```

```
a=log(x);
```

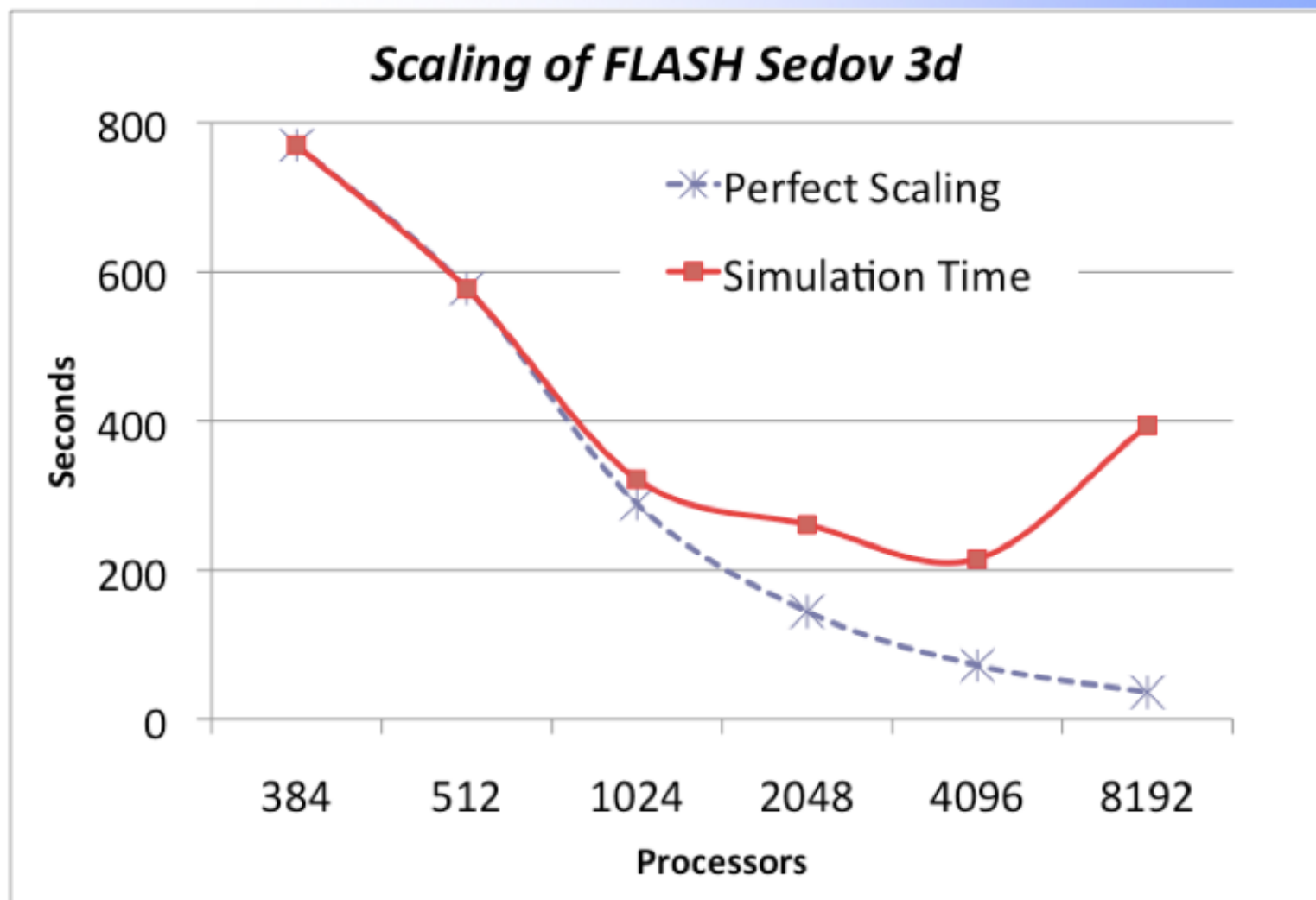
```
a=pow(x,5.);  
a=x*x*x*x*x;
```

```
a=i/sqrt(pow(sin(x),2.000001)+2.*pow(cos(x),2.000001));  
a=i*pow(1.+cos(x)*cos(x),-0.5);
```





FLASH Sedov 3d problem with Particles



Load Balance: real code

