560.602 GPU/CPU Programming for Engineers

(http://www.ce.jhu.edu/dalrymple/classes/602)

Prof. Robert A. Dalrymple
rad@jhu.edu
201 Latrobe Hall
410 516 7923

Fall, 2014

Meeting Time:	T Th 4:30-5:45
Location:	Krieger 307

Goal:
You will learn to write C++ computer programs. You will also learn to write them, using the full capability of multi-core CPUs and Nvidia GPUs for large computational speed-ups. Tools you will use include OpenMP for CPU programming, and CUDA for GPU programs.

Grading:

Homework/Classwork	20%
Midterm Exam	30
Programming Project	25
Final Exam	25

Topics

- C++ Programming
- OpenMP--multi-core parallel programming
- CUDA--GPU Programming
- CUDA Thrust Programming
- Using CUDA libraries

Web Resources and References

- C++ Language
- C++ Overview
- **Thinking in C++**, 2nd Ed., Vols 1 & 2, Bruce Eckel
- C++ Library Reference
- Introduction to Parallel Computing, Blaise Barney
- POSIX Threads Programming, Blaise Barney
- Guide to OpenMP: Easy Multithreading Programming for C++, Joel Yliluoma
- Intel: Getting Started with OpenMP, Richard Gerber
- Intel: More Work-Sharing with OpenMP, Richard Gerber
- Intel: Advanced OpenMP Programming, Richard Gerber
- GNU libgomp, the GNU OpenMP libary
- An Easy Introduction to CUDA C and C++, Mark Harris
- Nvidia CUDA Documentation and Guides

Programming Project: Options

Convert an existing program to parallel

Write a new research program for GPU

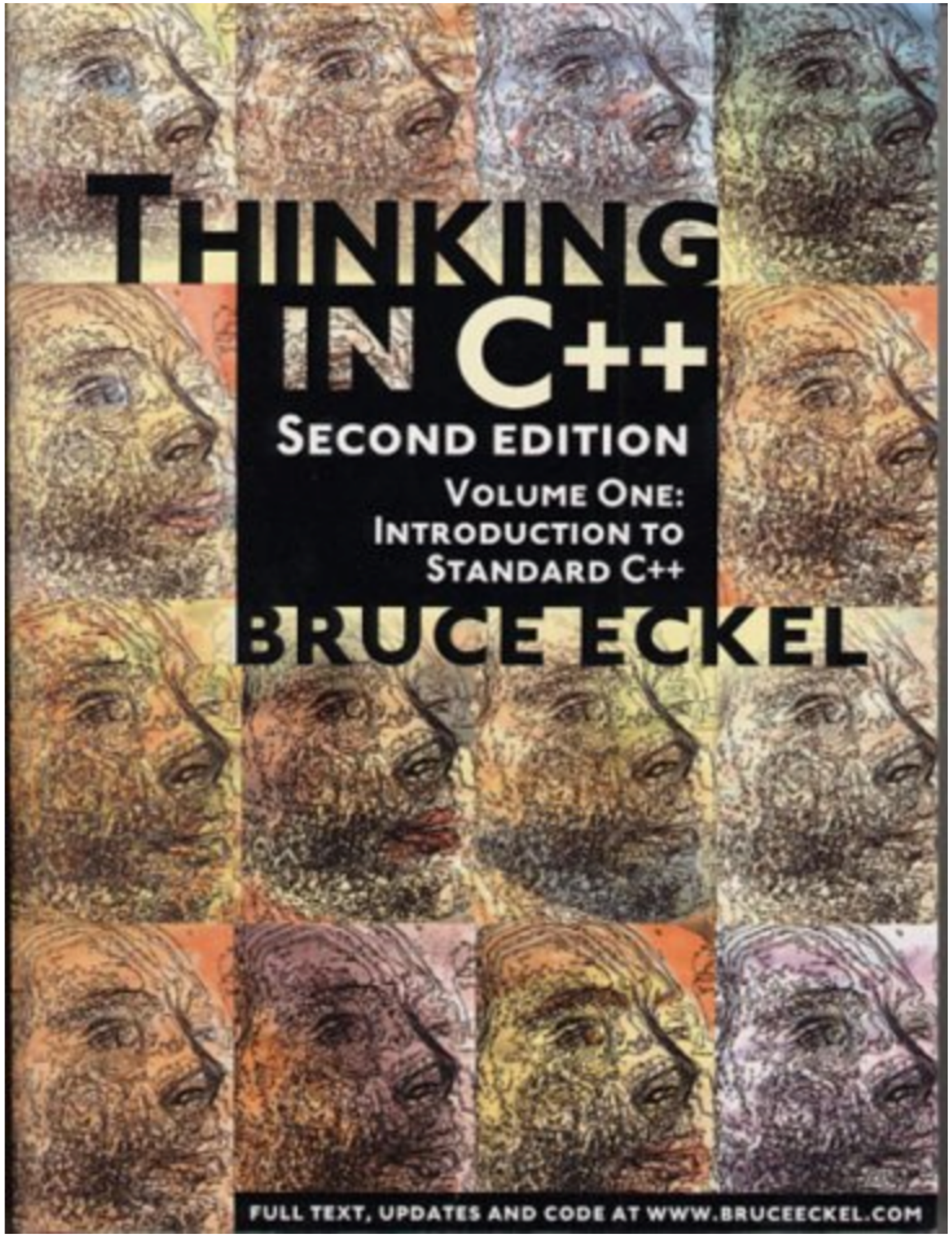
Write a new GPU program for me

<u>Project Write up:</u>

- The problem
- The methodology you used
- Optimizations used
- Speed-ups obtained

<u>Project Presentation:</u>

- In-class
- 15 minutes



Ebook: download html and code

Start reading->chap 3

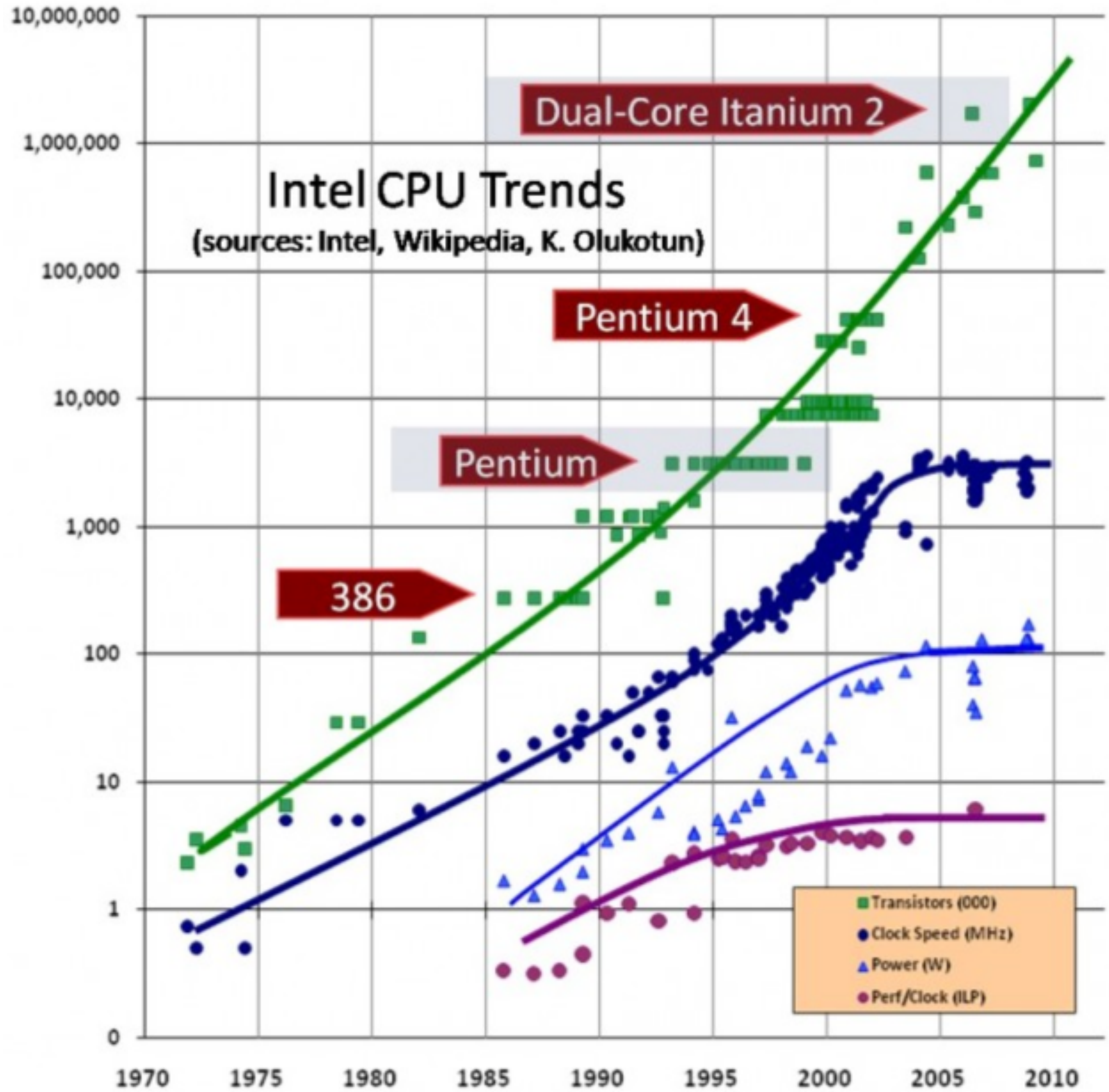
http://mindview.net/Books/TICPP/ThinkingInCPP2e.html

Need for parallel programming

Scientific/engineering problems can require immense computation: computation fluid dynamics, climate, meteorology, ocean circulation

Data sets are becoming much larger than computers

CPU



Moore's Law

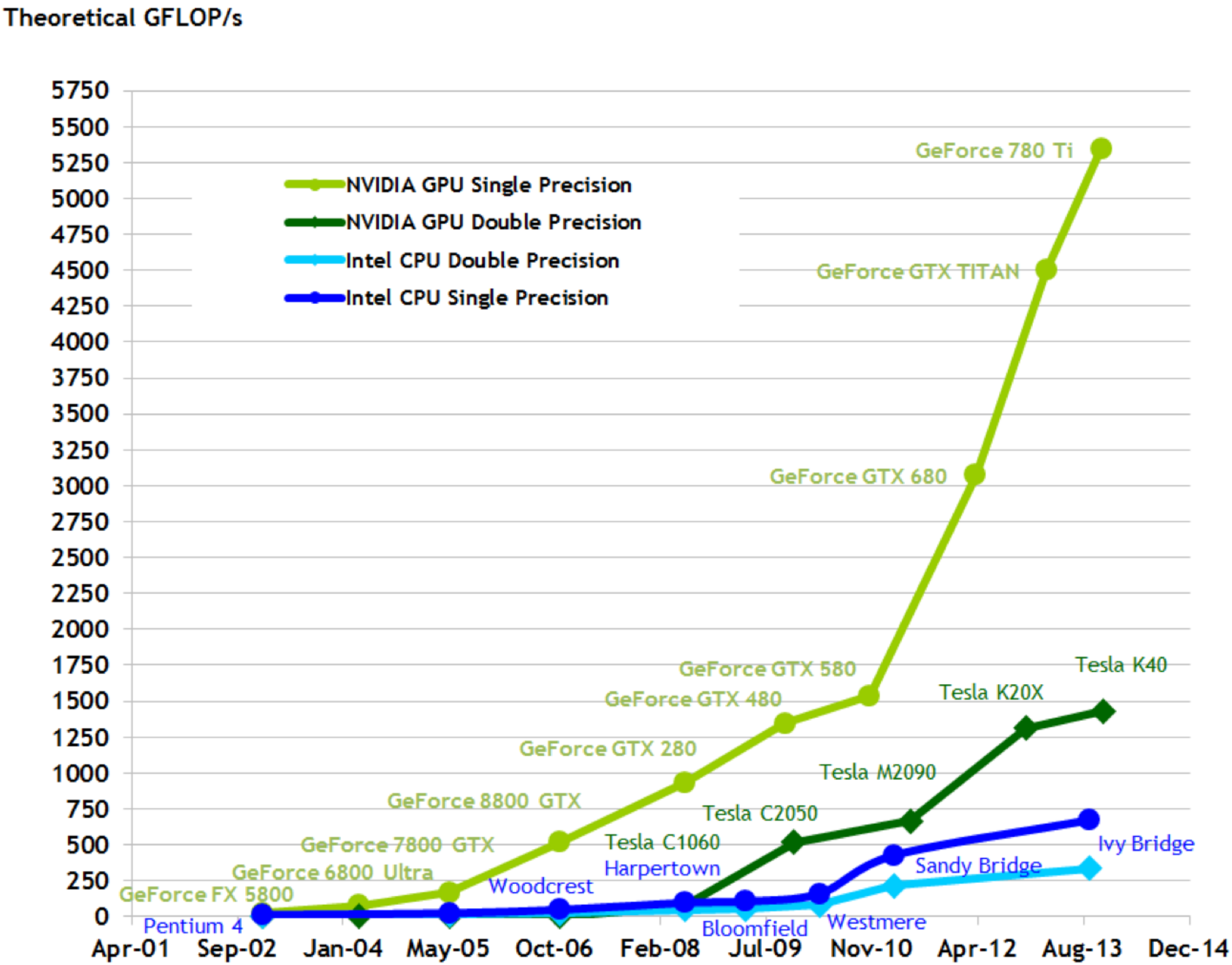
Clock Speed

Power

Performance

CPU scaling showing transistor density, power consumption, and efficiency. Chart originally from The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software

CUDA C Programming Guide



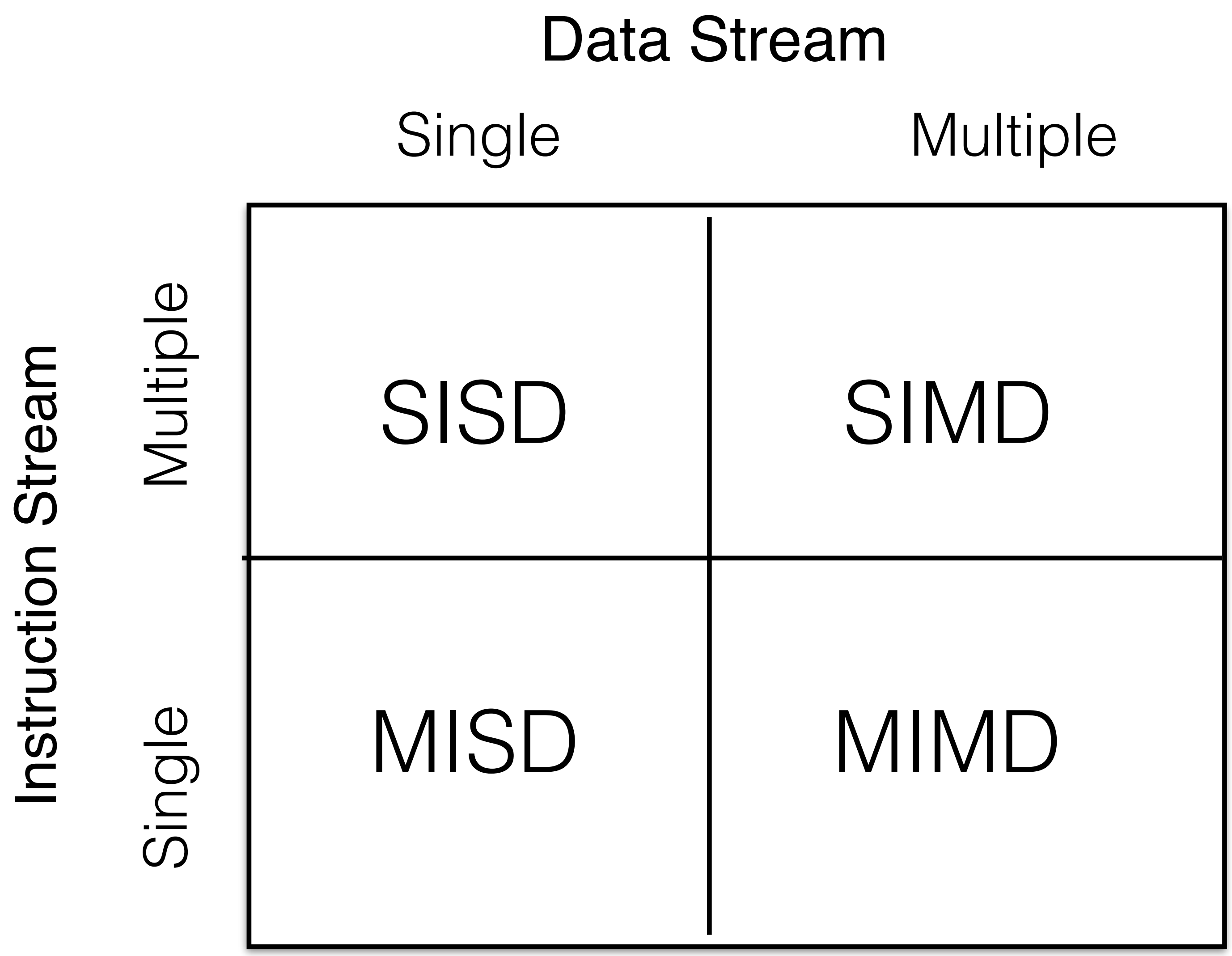
Computer performance

Name	FLOPS
yottaFLOPS	10^{24}
zettaFLOPS	10^{21}
exaFLOPS	10^{18}
petaFLOPS	10^{15}
teraFLOPS	10^{12}
gigaFLOPS	10^{9}
megaFLOPS	10^{6}
kiloFLOPS	10^{3}

CPU vs GPU

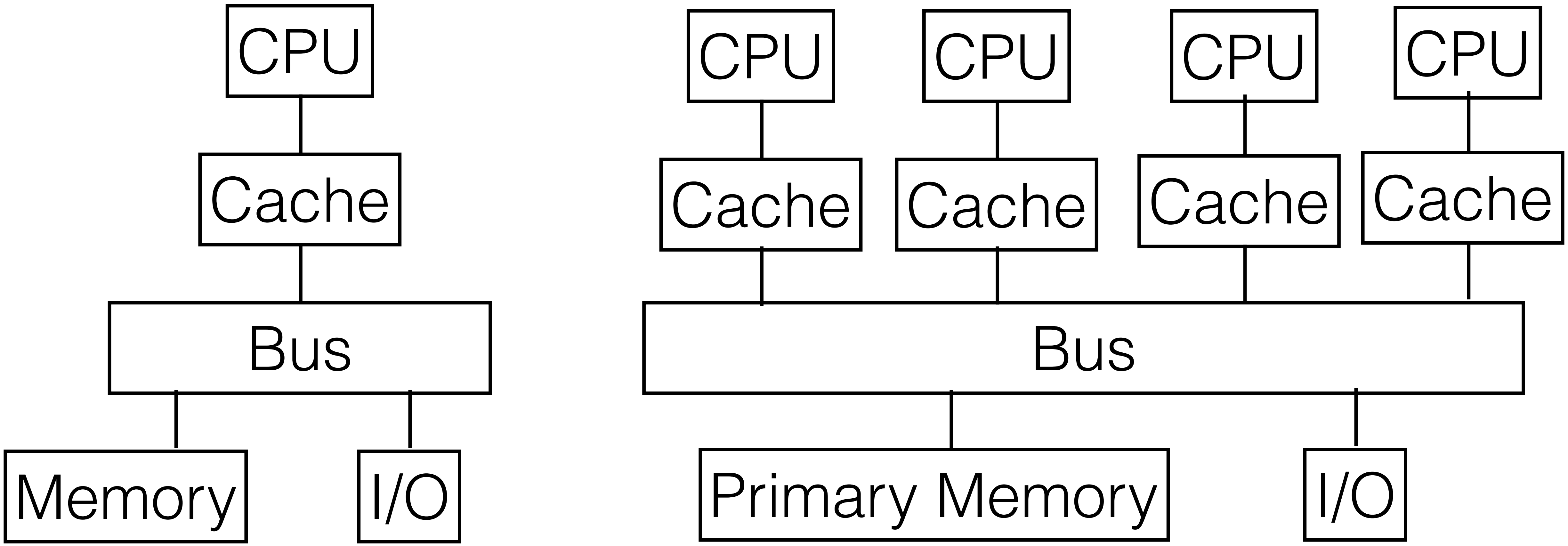
(de acuerdo a Adam y Jamie)

Flynn's Taxonomy



Flynn, 1996

CPU vs Multiprocessor



Centralized Multiprocessor

Different Approaches to Parallel Programming

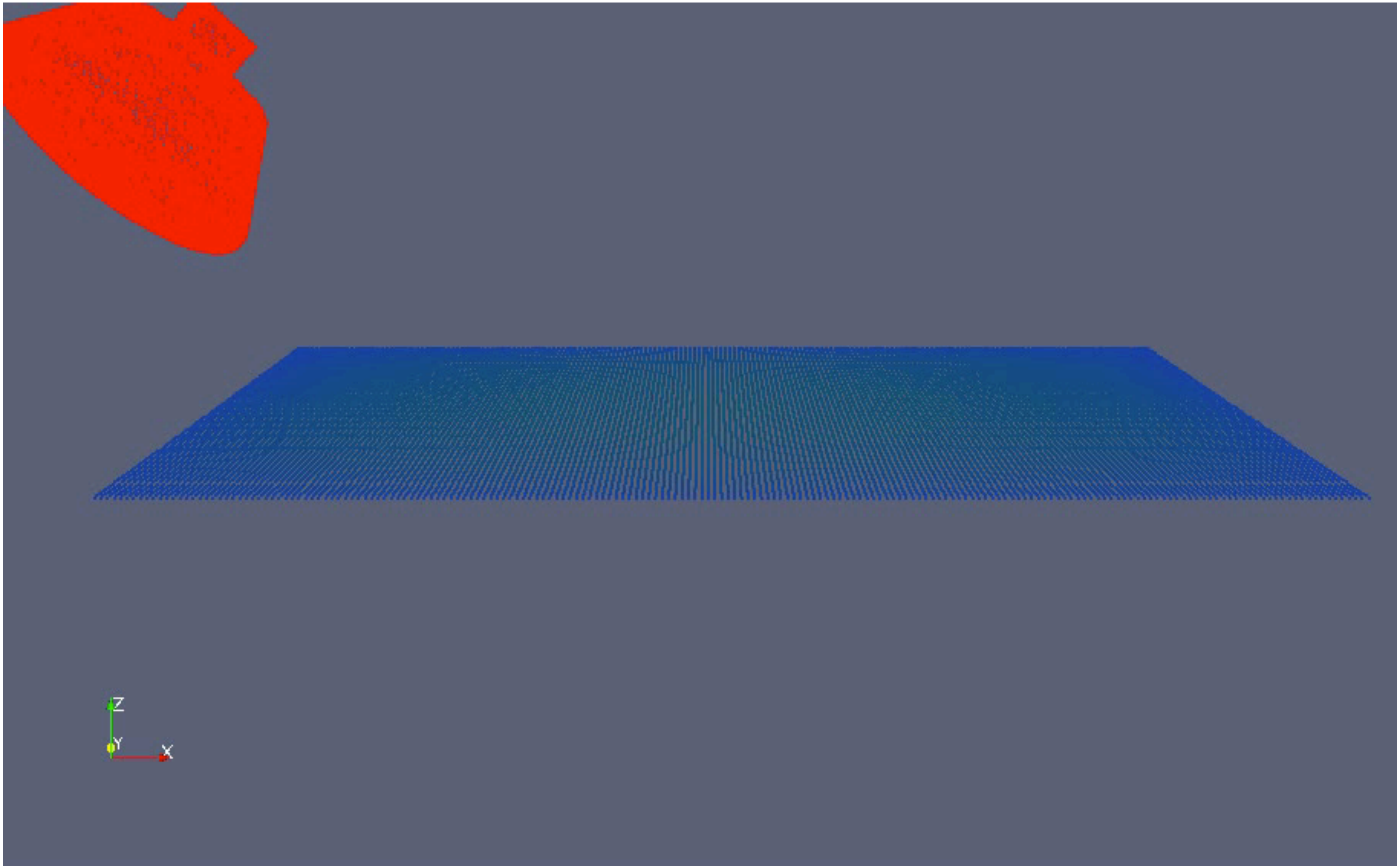
CPU

- Threads
- OpenMP
- OpenMPI

GPU

- CUDA C++, C, Fortran, Python (PyCUDA)
- Thrust
- Thrust/OpenMP
- ACC (new)

GPUSPH: GPU computing



Example: Compute Pi

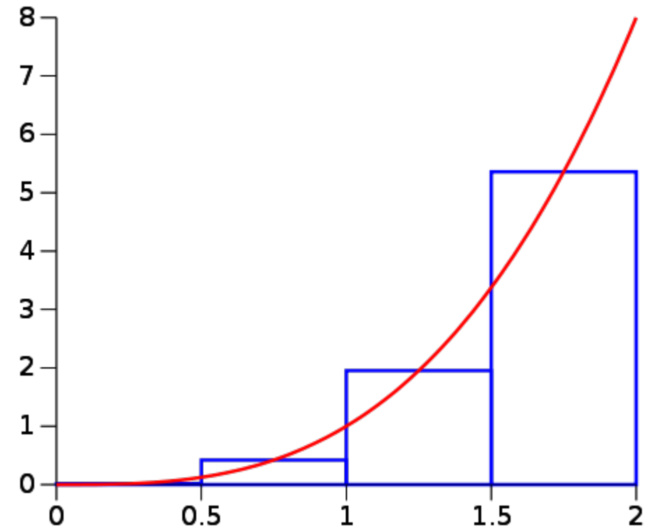
$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi$$

which is digitized to a summation over N intervals

$$w \sum_{i=0}^{N-1} \left(\frac{4}{1+(i+w/2)^2} \right) \text{ where } w = 1/N$$

Rectangular (Midpoint) Rule

Divide the interval into N equal pieces, *h= (b-a)/N*



Evaluate area of rectangle for each piece and sum

$$\int_a^b f(x)dx = h \sum_{i=0,N-1} f(a + (i + \frac{1}{2}h))$$

The bigger N, the smaller the interval and the better answer

```
#include <stdio.h>
#include  <stdlib.h> //this library includes atol
#include  <time.h>
#include <sys/time.h>
double getTimeElapsed(struct timeval end, struct timeval start)
   {
       return (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1000000.00;
   }

int main (int argc, char *argv[])
 {
  double PI25D= 3.141592653589793238462643;
  double x, w, totaltime, pi;
  int i, N;

 // timing info variables
  struct timeval t0, t1;
  double htime;

 pi=0.0;

 N=512;

 if (argc >1)   //read in the Number of intervals
{
  N= atol(argv[1]);  //parse command line for number of intervals

  printf("Number of intervals: %d \n",N);
  }
  else
   {
   printf("be sure you had an arg on command line for N greater than 512!\n");
   printf("format:  N of intervals \n Default is now 512\n");
   }
```

```
        w = 1.0f/((double)N);  // width of each computational segment
        printf(" w = %f \n”,w);

 gettimeofday(&t0, NULL);

  for (i=0; i<N ; i++)
   {
      x = (i+0.5)*w;
      pi += 4.0/( 1.0 + x*x);
    }


 gettimeofday(&t1, NULL);

 htime=getTimeElapsed(t1,t0);
 printf("time for parallel computation: %f \n", htime);
 pi *= w;
 printf("pi approx. %1.20f; error = %g \n", pi, PI25D-pi);
}
```

Examples for Pi: Computation Time

Program	N=100	10000	10,000,000
MyPinC	0.0000015	0.00010	0.047
MyPiOpenMP	0.00028	0.00027	0.013
MyPiGPU*	0.000031	0.00035	0.0004
ThrustPi *	0.00012	0.00033	0.0055

* GPU library

Consider No. of Processors

myPiOpenMP 100,000,000 intervals

No. of Processors	Time
2	0.207
4	0.15
6	0.126
8	0.112

Not a linear relationship between # processors and time

What you need

Computer
Text Editor
C++ compiler with OpenMP
CUDA nvcc compiler
    and video driver

Mac OS X

Dock: Applications

Finder (on Dock): Directories and Files
See directory: Applications

TextWrangler (on Dock): Text Editor

Terminal: (on Dock?): in Applications/Utilities

C++ compiler: /usr/local/bin/g++ fname

Simple UNIX commands

pwd	present working directory
cd name	change directory to name
cd ..	change directory up one level
rm name	remove file named name
ls	list files in directory
ls -l	list files with more information
mv fname fname2	change name
cp fname fname1	copy fname to fname1
cat fname	prints out contents of fname
man unix-command	gives help

Your first C++ program

Make a directory: CPP
Open TextWranger

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World";
    return 0;
}
```

Save in CPP as “hello.cpp”
In Terminal: cd CPP
                        /usr/local/bin/g++ hello.cpp
Type ./a.out