Project #4: Multi-Core Performance Evaluation & Analysis

18-640, Foundations of Compute Architecture, Fall 2015

Outline

- What is a compute stick?
- How to use a compute stick?
- Multi-Core Execution Libraries
- Image Processing Library & Edge Detection Example
- Performance Evaluation using Intel’s VTune
- Digit Detection
- Limitations of Compute Stick
- What you have to do?
- References

Outline

- What is a compute stick?
- How to use a compute stick?
- Multi-Core Execution Libraries
- Image Processing Library & Edge Detection Example
- Performance Evaluation using Intel's VTune
- Digit Detection
- Limitations of Compute Stick
- What you have to do?
- References

Intel’s Compute Stick

Peripherals:
- USB 2.0 port
- Power port
- HDMI
- Micro SD Card (not needed for our purpose)

Configuration:
- Quad-core Intel Atom Processor
- Windows 8.1
- 32 GB storage
- 2 GB memory
- Wifi card

How to use it?

- HDMI to DVI Adaptor (optional)

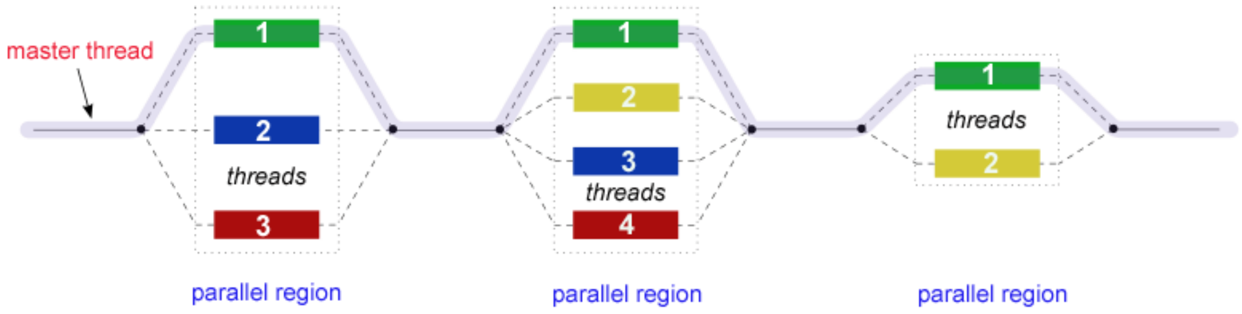
- USB Hub for external keyboard and mouse

Outline

- What is a compute stick?
- How to use compute stick?
- Multi-Core Execution Libraries
- Image Processing Library & Edge Detection Example
- Performance Evaluation using Intel's VTune
- Digit Detection
- Limitations of Compute Stick
- What you have to do?
- References

OpenMP

- Stands for Open specifications for MultiProcessing
- First released in 1997 for Fortran; released for C++ the following year
- Supports thread-based parallelism
- Uses fork-join model (may or may not support nested parallelism):



- May differ for each vendor
- Code simplicity, modularity, and scalability over Do-It-Yourself multithreads

Example: Compute the nth fibonacci number

Pthread version:

```
int fib(int n)
{
  if (n < 2) return n;
  else {
    int x = fib(n-1);
    int y = fib(n-2);
    return x + y;
  }
}
typedef struct {
  int input;
  int output;
} thread_args;

void *thread_func ( void *ptr )
{
  int i = ((thread_args *) ptr)->input;
  ((thread_args *) ptr)->output = fib(i);
  return NULL;
}
int main(int argc, char *argv[])
{
  pthread_t thread;
  thread_args args;
  int status, result;
  int thread_result;
  int n = atoi(argv[1]);
  if (n < 30) result = fib(n);
  else {
    args.input = n-1;
    status = pthread_create(&thread,  NULL, thread_func, (void*) &args );
    result = fib(n-2);
    pthread_join(thread, NULL);
    result += args.output;
  }
  printf("Fibonacci of %d is %d.\n", n, result);
  return 0;
}
```

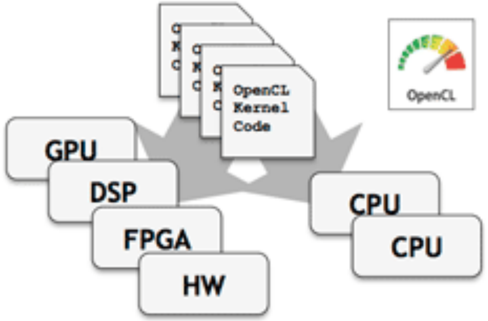
OpenMP version:

```
int fib(int n)
{
  if (n < 2) return n;
  else {
    int x = fib(n-1);
    int y = fib(n-2);
    return x + y;
  }
}
int main(int argc, char *argv[])
{
  int result;
  int n = atoi(argv[1]);
  if (n < 30) result = fib(n);
  else {
    int i,j;
#pragma omp task shared(i)
    i = fib(n-1);
#pragma omp task shared(j)
    j = fib(n-2);
#pragma omp taskwait
    result = i + j;
  }
  printf("Fibonacci of %d is %d.\n", n, result);
  return 0;
}
```

Advantage of OpenMP: Easy to read, modular, sclable; also less overhead.

OpenCL

- Publically released in December, 2008 by Khronos Group
- Open standard for developing cross-platform, vendor agnostic, parallel programs that run on current and future multi-core processors



- In comparison to OpenMP, OpenCL will have some extra overhead of compiling the kernel at runtime

Outline

- What is a compute stick?
- How to use compute stick?
- Multi-Core Execution Libraries
- Image Processing Library & Edge Detection Example
- Performance Evaluation using Intel's VTune
- Digit Detection
- Limitations of Compute Stick
- What you have to do?
- Questions?

OpenCV

- Open source BSD licensed computer vision library
- Started by Intel Research in 1998
- Written in C++ with bindings available for Python, Java, Matlab
- Some questions in Project 4 will be based on image processing

Image: How is it represented?

- Two dimensional array of pixels

- Binary image:
  - Pixels are bits

- Grayscale image:
  - Pixels are scalars

- Color image:
  - Pixels are vectors

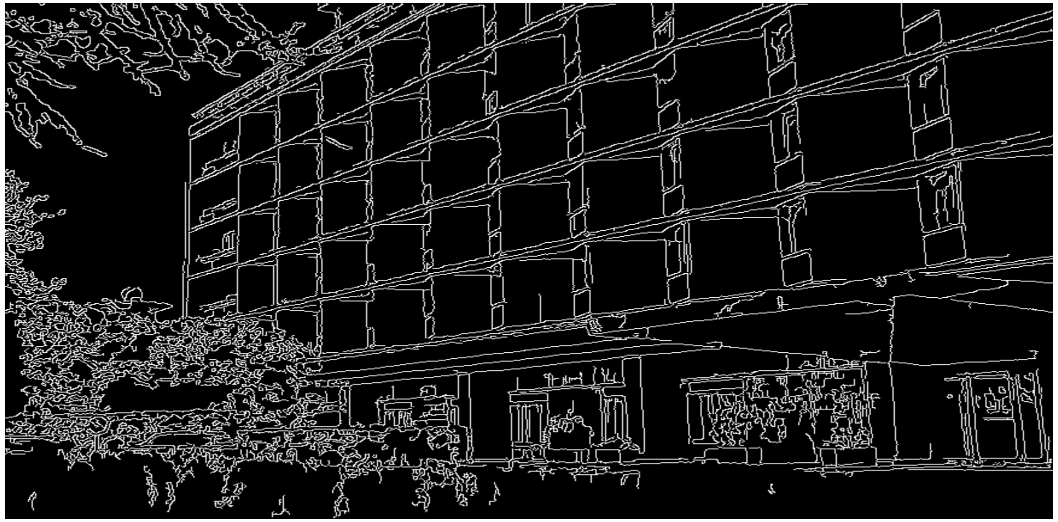
Index	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50
5	51	52	53	54	55	56	57	58	59	60
6	61	62	63	64	65	66	67	68	69	70
7	71	72	73	74	75	76	77	78	79	80
8	81	82	83	84	85	86	87	88	89	90
9	91	92	93	94	95	96	97	98	99	100

Grayscale Image in form of a matrix

Using OpenCV for Edge Detection

- Input image:

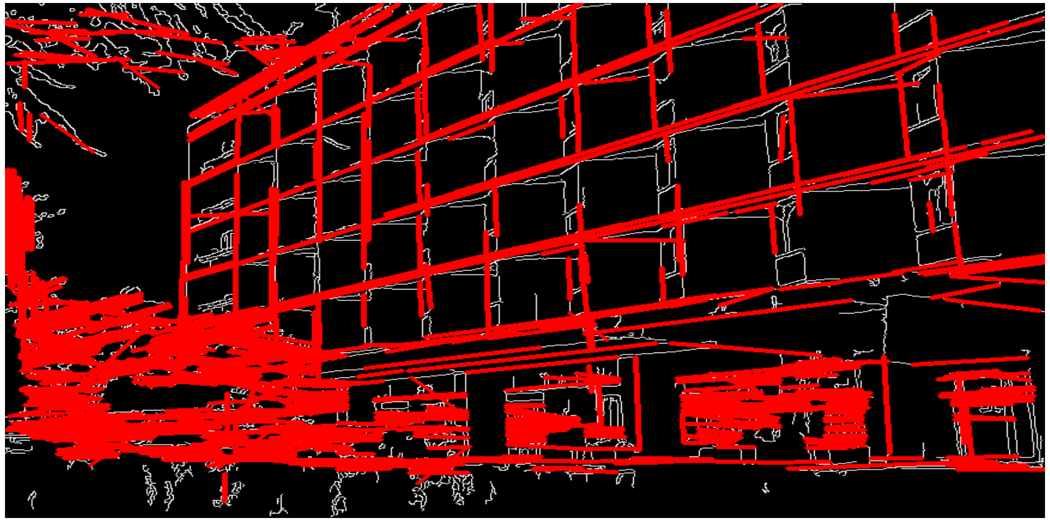
- Canny algorithm: for edge detection



Using OpenCV for Line Detection

- Input image:

- Canny + Hough transform: edge-detection and feature extraction



Outline

- What is a compute stick?
- How to use compute stick?
- Multi-Core Execution Libraries
- Image Processing Library & Edge Detection Example
- Performance Evaluation using Intel’s VTune
- Digit Detection
- Limitations of Compute Stick
- What you have to do?
- References

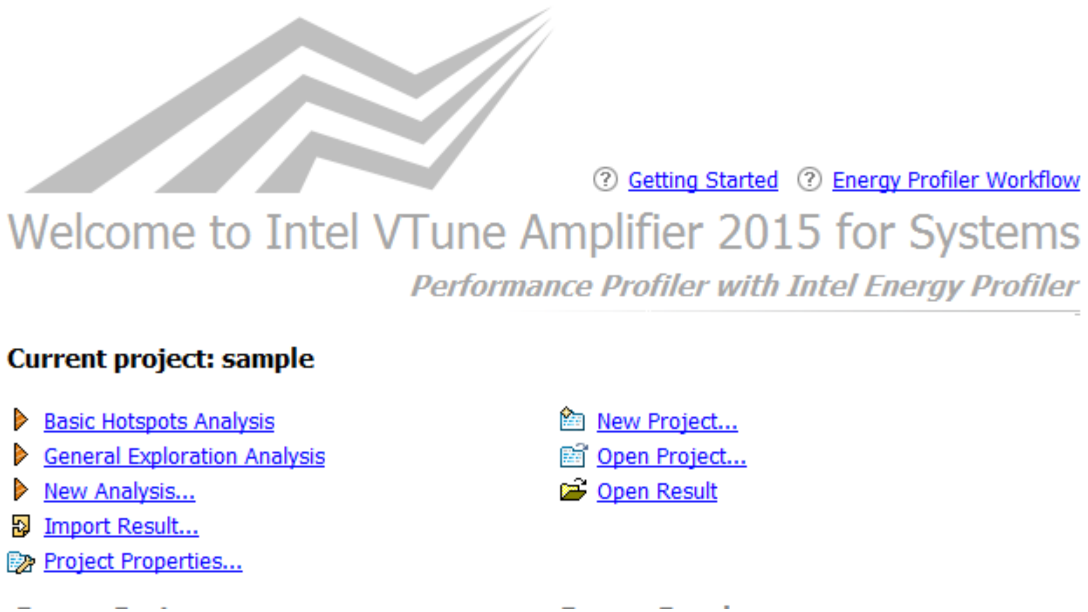
VTune Amplifier

- Helps in collections of performance data of an application running on the system

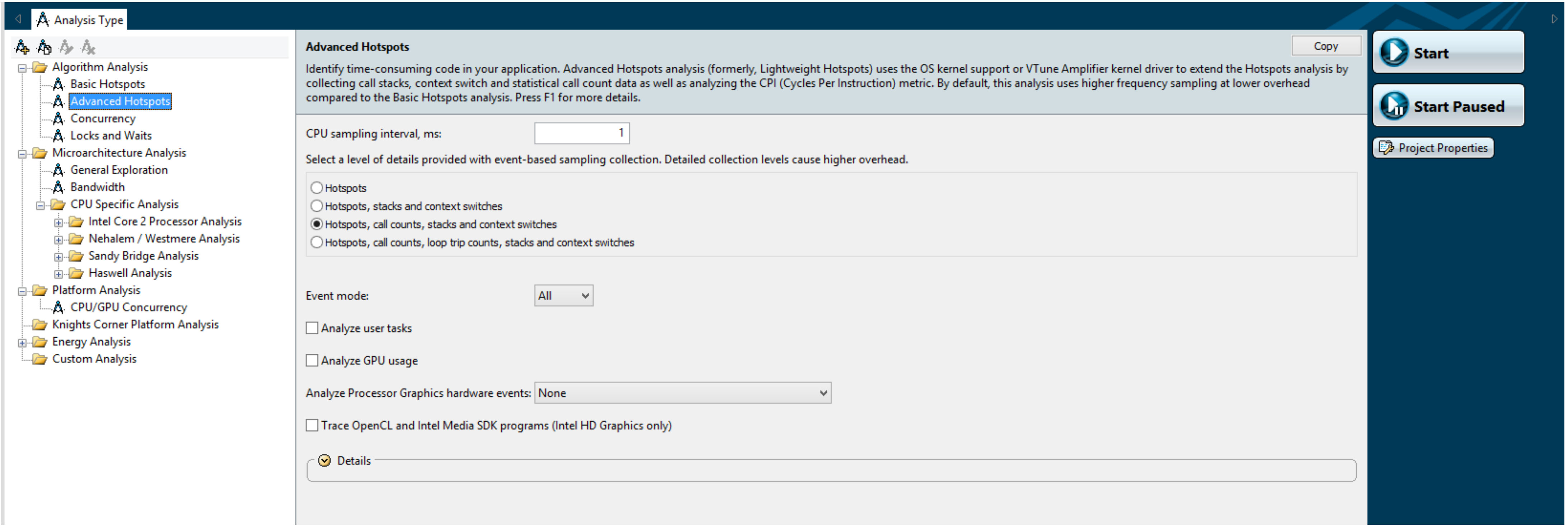
- Organizes data in different views

- Helps to identify potential issues and also suggests improvements

Vtune Example – Step 1: Create Project



Vtune Example – Step 2: Choose analysis type & start



VTune Example – Step 3: View Summary

Elapsed Time:	**53.202s**
CPU Time:	48.692s
Effective Time:	48.686s
Spin Time:	0.006s
Overhead Time:	0s
Instructions Retired:	41,371,428,877
Estimated Call Count:	46,382,814
CPI Rate:	2.065

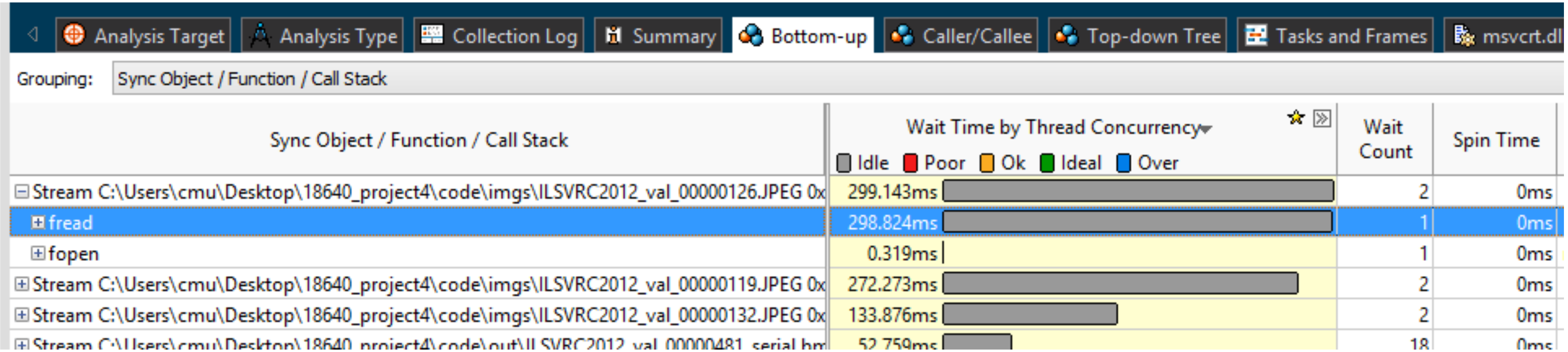
The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

Wait Rate:	5.262
CPU Frequency Ratio:	1.316
Context Switch Time:	2.460s
Wait Time:	0.226s
Inactive Time:	2.234s
Paused Time:	0.116s
Estimated Call Count:	46,382,814

VTune Example – Some terms

- CPU time: time for which CPU is actively executing
- Effective time: time spent in the user code (excludes spin and overhead time)
- Wait time: time while threads were waiting for synchronization
- Cycles per instruction (CPI): average amount of time in machine clock cycles for each instruction

Vtune Example – Step 4: Analyze other tabs



1. Click on function to get source code
2. From source code, one can also navigate to Instruction Manual (do install Adobe Reader)

Outline

- What is a compute stick?
- How to use compute stick?
- Multi-Core Execution Libraries
- Image Processing Library & Edge Detection Example
- Performance Evaluation using Intel's VTune
- Digit Detection
- Limitations of Compute Stick
- What you have to do?
- References

Digit Detection

- Collection of images where each image represents a digit between 0-9
- Every digit is represented by a 28x28 matrix
- Every cell value of the matrix represents a pixel with a discrete value between 0 - 255

Digit Detection Algorithm – Nearest Neighbor

1. Store a set of images (matrix and its label) in memory – *training set*
2. For every new image *i* in the *test set*:
   I. find nearest neighbor by calculating the euclidean distance from every image in the training set
   II. assign label of the training image which has the minimum distance to test image *i*

Outline

- What is a compute stick?
- How to use compute stick?
- Multi-Core Execution Libraries
- Image Processing Library & Edge Detection Example
- Performance Evaluation using Intel’s VTune
- Digit Detection
- Limitations of Compute Stick
- What you have to do?
- References

Limitations of Compute Stick

- Not ideal for tasks which are computationally heavy
- Low memory so Vtune may crash
- Some restriction on collection of GPU metrics due to BIOS
- RDP not possible with the pre-installed version of Windows (alternative TightVNC)

Outline

- What is a compute stick?
- How to use compute stick?
- Multi-Core Execution Libraries
- Image Processing Library & Edge Detection Example
- Performance Evaluation using Intel's VTune
- Digit Detection
- Limitations of Compute Stick
- What you have to do?
- References

What you have to do?

- Set up Windows on Compute stick
    - Set Windows username as **cmu_18640**
    - Set password as **root**

- Install the following:
    - MinGW (Minimalist GNU for Windows)
    - OpenCL
    - Intel VTune Amplifier

What you have to do?

- Add proper paths to the system path variables. For example:
  - If you run multi-threaded with SSE (mt_sse) make it has in it's path the muti-threaded with SSE libraries in its path
- Compile the program using ming32-make
- Run VTune (**as administrator**) with each executable and observe the reading for different types of analysis

Report

- Answers all questions under Report sections in the handout.

Matrix Multiplication	Edge Detection	Digit Detect
mmm_single_thread_scalar	st_nsse	st_digit
mmm_single_thread_simd	st_sse	
mmm_multi_thread_scalar	mt_nsse	
mmm_multi_thread_simd	mt_sse	
	mt_sse_ocl	

Yes, we have a bonus section too!

- Parallelize the st_digit (digit detection) code to make it run faster.
- Go through the sequential code and see how you can parallelize it.
- Use any method to make it parallel.
- Report the same parameters for the parallel version as asked in previous part.
- Score bonus 10 points!

Please return compute sticks in its original box no later than final exam day

References

- Lecture notes by Simon Lucey (http://16423.courses.cs.cmu.edu/slides/Lecture_2.pdf)
- OpenMP: A short introduction, 18-740, CMU Fall 2009 (http://www.cs.cmu.edu/afs/cs/academic/class/15740-f09/public/asst/asst2/openmp_tutorial.pdf)
- OpenCL Overview (http://sa10.idav.ucdavis.edu/docs/sa10-dg-opencl-overview.pdf)
- Mohamed Zahran, NYU (http://cs.nyu.edu/courses/fall12/CSCI-GA.3033-012/lecture7.pdf)
- LLNL pages for OpenMP and OpenCL
- https://www.khronos.org/opencl/

Questions