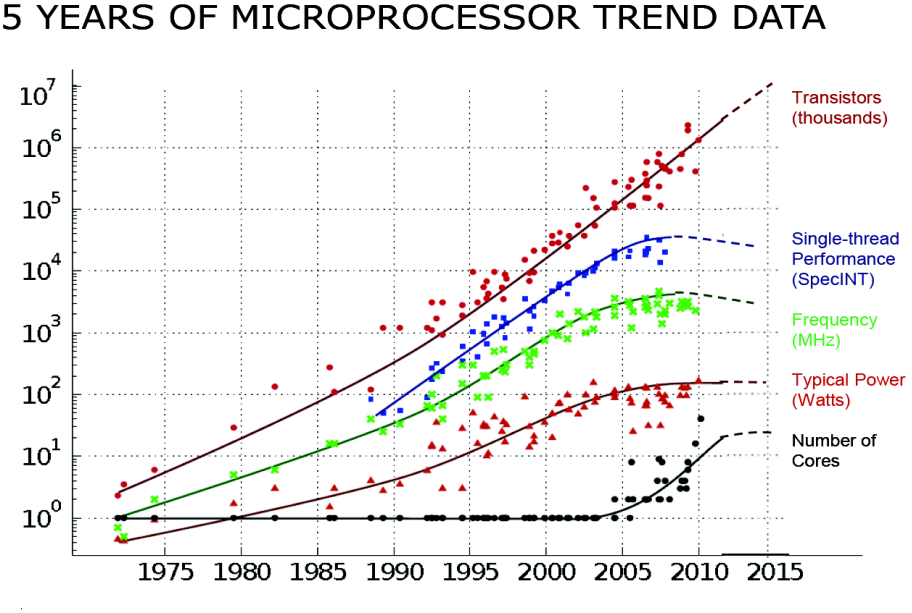
KTÜ
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü
Mikroişlemciler Laboratuvarı

Grafik İşlemcileri ile Hesaplama

1. Giriş

Intel Şirketinin kurucularından, Gordon Moore, 1965 yılındaki bir ifadesinde "*...Tümleşik devrelerde (işlemcilerde) birim alanda kullanılan transistor sayısı 2 katına çıkacaktır,*" hipotezi Intel şirketince ilke edinilmiştir. Sonraki yıllarda yapılan gözlemlerde bu ifade "*... kullanılan transistor sayısı 18 ayda 2 katına çıkacaktır.*" şeklinde güncellenmiştir. 35 yılı aşkın sürdürülen bu gelişme "Moore Yasası" olarak bilinmektedir.

2000'li yılların başında "ısı duvarı" (heat wall veya power wall) problemi ile karşılaşıncaya kadar Moore Yasasına sadık kalındı. Isı duvarı birim alanda kullanılan transistor sayısı arttıkça, doğal olarak, birim alana düşen güç tüketimi de artmıştır. Bu sayede yarı iletkenlerinin özelliği gereği açığa çıkan ısı katlanmıştır. Isınan işlemciler devreden uzaklaştırılamadığı için bu probleme "ısı duvarı" adı verilmiştir. Bu durum üretici firmaları 2004 yılından itibaren çok çekirdekli işlemciler üretmeye sevk etti. 1970'lerden itibaren geliştirilen işlemcilerin teknik özelliklerini özetleyen grafik aşağıda sunulmuştur.



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Karl Rupp "42 Years of Microprocessor Trend Data"

2. OpenCL

OpenCL, (*Open Computing Language*), 2008 yılında, Apple şirketinin Khronos Group'a teklifi ile Khronos Group tarafından geliştirilen paralel programlama platformudur. Açık kaynak bir sistemdir ve spesifikasyonu pek çok şirketin katkılarıyla geliştirilmeye devam edilmektedir. CPU ve GPU'nun (DSP ve FPGA'de olabilir) bir çatı altında programlanabildiği heterojen bir yapısı vardır. OpenCL AMD, Intel, ARM ve NVIDIA (son yıllarda NVIDIA desteğini çekmiş görünüyor) tarafından desteklenmektedir. Ayrıca IBM Cell işlemciler ile de hesaplama yapılabilmektedir.

- **Host:** OpenCL programını çalıştırılacağı CPU ve ana belleğin bulunduğu PC.

- **Device:** Host'a bağlı ve host tarafından kontrol edilen FPGA, ekran kartı gibi cihazlardır.
- **Çekirdek Fonksiyon (Kernel):** Cihaz üzerinde çalıştırılabilen fonksiyonlara verilen isimdir.

1. OpenCL'in Özellikleri

- Çoklu platformlarda çalışabilen hesaplama platformudur
- Çekirdek fonksiyonlar C99 standardını destekler ve eklemeler mevcuttur.
- Run-time derleme yapıllabilir.
- IEEE 754 reel sayı formatını destekler.
- Senkronizasyon ve çalışma modeline ilişkin pek çok fonksiyon eklenmiştir.
- Bellekteki veriye erişim için __global, __local, __constant, ve __private sıfatları eklenmiştir.

2. OpenCL' de Dizi Çarpma Örneği

n elemanlı iki dizinin elemanlarını C dilinde birebir çarpma işlemini aşağıdaki gibi bir fonksiyon ile gerçekleştirebiliriz. Bu işlemi OpenCL ile yapmak için ise paralel çalışacak çekirdek fonksiyonu yazılır. Bu kernel her iş-parçası yalnızca bir işlem gerçekleştirecektir.

Traditional loops

```
void
mul(const  int n,
    const  float *a,
    const  float *b,
           float *c)
{
  int i;
  for (i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

OpenCL

```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)
{
  int id = get_global_id(0);
  c[id]  = a[id] * b[id];
}
// execute over n work-items
```

3. OpenCL Programlama Adımları

1. OpenCL aygıtları taranır: Sistemde OpenCL'i destekleyen cihazlar var mı tespit edilir.
2. Kontext: OpenCL uygulamasının dış çerçevesi gibi düşünebiliriz.
3. Programlar oluşturulur: Kontext içinde bir veya daha fazla OpenCL cihazını kullanmak için,
4. Program içinden çekirdek fonksiyonları belirlenir.
5. Memory nesneleri ile bellekte gerekli yer ayırma ve kopyalama işlemlerini modellenir.
6. Gerekli veri aktarımları yapılır,
7. Çekirdek fonksiyonları "komut kuyruğuna" yerleştir
8. Programı çalıştırılır.

3. DENEYE HAZIRLIK

1. FLOP ve OP(eration) kavramlarını araştırınız
2. OpenCL nedir araştırınız. CUDA ile aralarındaki farkları belirtiniz.
   a. OpenCL'in özelliklerini listeleyiniz
3. Flynn taksonomisini inceleyiniz.
   a. Dağıtık ve ortak bellekli sistemlere örnekler veriniz.
4. Amdahl yasasının özellikleri nelerdir araştırınız.
   a. Speedup, efficiency gibi kavramların ne anlama geldiğini açıklayınız.
5. Bulabildiğiniz en gelişmiş CPU ve GPU işlemcilerini (mümkünse birkaç tanesini) karşılaştırınız.
6. Aşağıda verilen listeyi inceleyiniz ve ne olabileceği hakkında fikir yürütünüz.

CL_DEVICE_VENDOR_ID:	32902
CL_DEVICE_AVAILABLE:	1
CL_DEVICE_VENDOR:	Intel(R) Corporation
CL_DEVICE_NAME:	Intel(R) HD Graphics 4400

```
CL_DRIVER_VERSION:                      10.18.14.4264
CL_DEVICE_ADDRESS_BITS:                 64
CL_DEVICE_ENDIAN_LITTLE:                1
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE:        262144
CL_DEVICE_LOCAL_MEM_SIZE:               65536
CL_DEVICE_MAX_CLOCK_FREQUENCY:          400
CL_DEVICE_MAX_COMPUTE_UNITS:            20
```

4. **DENEYİN YAPILIŞI**

Önceden hazırlanmış bir program üzerinde amaca uygun değişiklikler ve eklemeleri yapmanız istenmektedir.
Hazırlanan uygulamanın çıkan sonuçları yorumlayınız. Parametrelerin değişiklikleri yapılarak çalıştırılan programın sonuçlarını irdeleyiniz.

Ek: Örnek Uygulama

```
/* OpenCL örneği*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <CL/cl.h>

// OpenCL kernel. Each work item takes care of one element of c
const char *kernelSource = "\n"                                    \
"#pragma OPENCL EXTENSION cl_khr_fp64 : enable \n"    \
"__kernel void vecAdd(  __global double *a,                 \n"  \
"                       __global double *b,                 \n"  \
"                       __global double *c,                 \n"  \
"                       const unsigned int n)               \n"  \
"{                                                          \n"  \
"    //Get our global thread ID                             \n"  \
"    int id = get_global_id(0);                             \n"  \
"                                                           \n"  \
"    //Make sure we do not go out of bounds                 \n"  \
"    if (id < n)                                            \n"  \
"        c[id] = a[id] + b[id];                             \n"  \
"}; "

int main(int argc, char* argv[]){
        unsigned int n = 100000; // Length of vectors
        double *h_a, *h_b;  // Host input vectors
        double *h_c;        // Host output vector

        cl_mem d_a, d_b;   // Device input buffer
        cl_mem d_c;        // Device output buffer

        cl_platform_id cpPlatform;      // OpenCL platform
        cl_device_id device_id;         // device ID
        cl_context context;             // context
        cl_command_queue queue;   // command queue
        cl_program program;             // program
        cl_kernel kernel;               // kernel

        // Size, in bytes, of each vector
        size_t bytes = n*sizeof(double);

        // Allocate memory for each vector on host
        h_a = (double*)malloc(bytes);
        h_b = (double*)malloc(bytes);
        h_c = (double*)malloc(bytes);

        // Initialize vectors on host
        int i;
        for (i = 0; i < n; i++){
                h_a[i] = sinf(i)*sinf(i);
                h_b[i] = cosf(i)*cosf(i);
        }

        size_t globalSize, localSize;
```

```
	cl_int err;
// Number of work items in each local work group
	localSize = 64;
// Number of total work items - localSize must be devisor
	globalSize = ceil(n / (float)localSize)*localSize;
// Bind to platform
	err = clGetPlatformIDs(1, &cpPlatform, NULL);
// Get ID for the device
	err = clGetDeviceIDs(cpPlatform, CL_DEVICE_TYPE_GPU, 1, &device_id, NULL);
// Create a context
	context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);
// Create a command queue
	queue = clCreateCommandQueue(context, device_id, 0, &err);
// Create the compute program from the source buffer
	program = clCreateProgramWithSource(context, 1,
		(const char **)& kernelSource, NULL, &err);
// Build the program executable
	clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
// Create the compute kernel in the program we wish to run
	kernel = clCreateKernel(program, "vecAdd", &err);

	// Create the input and output arrays in device memory for our calculation
	d_a = clCreateBuffer(context, CL_MEM_READ_ONLY, bytes, NULL, NULL);
	d_b = clCreateBuffer(context, CL_MEM_READ_ONLY, bytes, NULL, NULL);
	d_c = clCreateBuffer(context, CL_MEM_WRITE_ONLY, bytes, NULL, NULL);

	// Write our data set into the input array in device memory
	err = clEnqueueWriteBuffer(queue, d_a, CL_TRUE, 0, bytes, h_a, 0, NULL, NULL);
	err |= clEnqueueWriteBuffer(queue, d_b, CL_TRUE, 0, bytes, h_b, 0, NULL, NULL);

	// Set the arguments to our compute kernel
	err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &d_a);
	err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &d_b);
	err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &d_c);
	err |= clSetKernelArg(kernel, 3, sizeof(unsigned int), &n);

	// Execute the kernel over the entire range of the data set
	err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &globalSize, &localSize,
		0, NULL, NULL);

	// Wait for the command queue to get serviced before reading back results
	clFinish(queue);

	// Read the results from the device
	clEnqueueReadBuffer(queue, d_c, CL_TRUE, 0, bytes, h_c, 0, NULL, NULL);

	//Sum up vector c and print result divided by n, this should equal 1 within error
	double sum = 0;
	for (i = 0; i<n; i++) sum += h_c[i];
	printf("final result: %f\n", sum / n);
	// release OpenCL resources
	clReleaseMemObject(d_a);
	clReleaseMemObject(d_b);
	clReleaseMemObject(d_c);
	clReleaseProgram(program);
	clReleaseKernel(kernel);
	clReleaseCommandQueue(queue);
	clReleaseContext(context);
	//release host memory
	free(h_a); free(h_b); free(h_c);
	return 0;
}
```