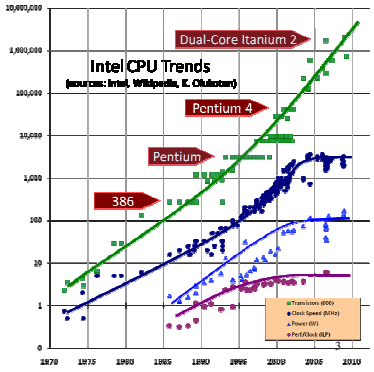
Advanced Architecture Topics

CPU History

- Intel CPU history
  - → CPU trends
  - → Microprocessor designs
- GPUs
  - → Handle specialized libraries & algorithms
  - → Special programming interface needed
- Power
  - → Reducing power usage

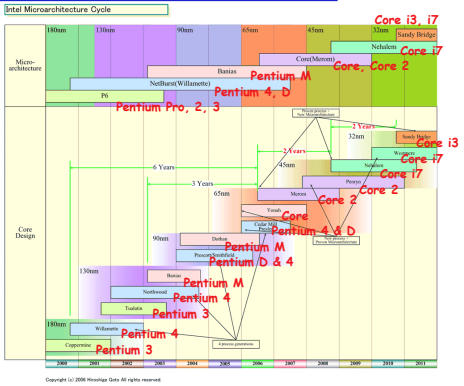
Intel CPU Trends

- # transistors keeps increasing
- Clock speed plateaus at around 2-3 Ghz around 2003
- Power use plateaus at 100 watts around 2003
- ILP (instructions per cycle) plateaus at around 8/cycle by 2000



Intel Microprocessor Designs

- Micro-architecture designs overlap in time
- Semi-conductor technology continues improving
- Fabrication feature width drops from 180nm to 32nm
- 22nm 3D Tri-Gate announced 2011



Graphics Processing Units (GPUs)

- CPUs
- Lots of instructions little data
  - Out of order exec
  - Branch prediction
- Reuse and locality
- Task parallel
- Needs OS
- Complex sync
- Latency machines

- GPUs
- Few instructions lots of data
  - SIMD
  - Hardware threading
- Little reuse
- Data parallel
- No OS
- Simple sync
- Throughput machines

Programming GPUs

- GPGPU
  - → General purpose computing on GPUs
    - Using special libraries (e.g. CUDA) to copy / process data
- Approach
  - → GPUs can compute vector / stream operations in parallel
    - Requires programs for both CPU & GPU
  - → Compiler can simplify process of generating GPU code
    - PGI compiler relies on user-inserted annotations to specify parallel region, vector operations
- Advantages
  - → Supercomputer-like FP performance on commodity processors
- Disadvantages
  - → Performance tuning difficult
  - → Large speed gap between compiler-generated and hand-tuned code

Matrix Multiplication Example

- Original Fortran

```
do i = 1,n
  do j = 1,m
    do k = 1,p
      a(i,j) = a(i,j) + b(i,k)*c(k,j)
    enddo
  enddo
enddo
```


Matrix Multiplication Example

- Hand-written GPU code using CUDA

```
__global__ void
matmulKernel( float* C, float* A, float* B, int N2, int N3 ){
  int bx = blockIdx.x,  by = blockIdx.y;
  int tx = threadIdx.x, ty = threadIdx.y;
  int aFirst = 16 * by * N2;
  int bFirst = 16 * bx;
  float Csub = 0;

  for( int j = 0; j < N2; j += 16 ) {
    __shared__ float Atile[16][16], Btile[16][16];
    Atile[ty][tx] = A[aFirst + j + N2 * ty + tx];
    Btile[ty][tx] = B[bFirst + j*N3 + b + N3 * ty + tx];

    __syncthreads();

    for( int k = 0; k < 16; ++k )
      Csub += Atile[ty][k] * Btile[k][tx];

    __syncthreads();
  }

  int c = N3 * 16 * by + 16 * bx;
  C[c + N3 * ty + tx] = Csub;
}
```


Matrix Multiplication Example

- Hand-written CPU code using CUDA

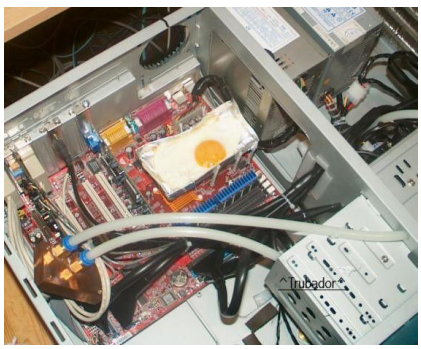
```
void
matmul( float* A, float* B, float* C,
        size_t N1, size_t N2, size_t N3 ){
  void *devA, *devB, *devC;
  cudaSetDevice(0);

  cudaMalloc( &devA, N1*N2*sizeof(float) );
  cudaMalloc( &devB, N2*N3*sizeof(float) );
  cudaMalloc( &devC, N1*N3*sizeof(float) );

  cudaMemcpy( devA, A, N1*N2*sizeof(float), cudaMemcpyHostToDevice );
  cudaMemcpy( devB, B, N2*N3*sizeof(float), cudaMemcpyHostToDevice );

  dim3 threads( 16, 16 );
  dim3 grid( N1 / threads.x, N3 / threads.y);

  matmulKernel<<< grid, threads >>>( devC, devA, devB, N2, N3 );

  cudaMemcpy( C, devC, N1*N3*sizeof(float), cudaMemcpyDeviceToHost );
  cudaFree( devA );
  cudaFree( devB );
  cudaFree( devC );
}
```


Matrix Multiplication Example

- Annotated Fortran for PGI compiler (compiled to CUDA)

```
!$acc region
  !$acc do parallel
  do j=1,m
    do k=1,p
      !$acc do parallel, vector(2)
    do i=1,n
      a(i,j) = a(i,j) + b(i,k)*c(k,j)
    enddo
    enddo
  enddo
!$acc end region
```


Power-Aware Computing

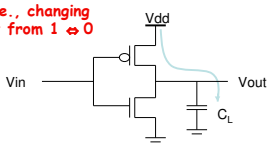


Cooking an egg on the CPU

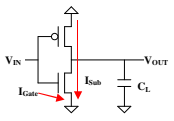
Power Issues in Microprocessors

Capacitive (Dynamic) Power

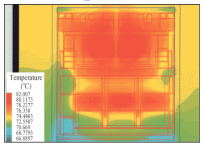
I.e., changing bit from 1 ⇔ 0



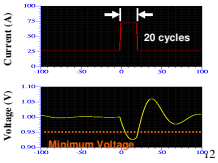
Static (Leakage) Power



Temperature



Di/Dt (Vdd/Gnd Bounce)



Architecture Features for Low Power

- Voltage scaling
  - Reduce voltage to save power when performance not needed
- Processor-specific instruction sets
  - On ARM the default int type is ~ 20% more efficient than char or short (sign/zero extension)
- Processor-specific features
  - Shut off unused registers to save power