Computational Science on Many-Core Architectures

360.252

Karl Rupp

Institute for Microelectronics, TU Wien
http://www.iue.tuwien.ac.at/

Zoom Channel 941 8518 8102
Q&A on Wednesday, October 12, 2022

CUDA

About

- Initial release in 2007
- Proprietary programming model by NVIDIA
- C++ with extensions
- Proprietary compiler extracts GPU kernels

Software Ecosystem

- Vendor-tuned libraries: cuBLAS, cuSparse, cuSolver, cuFFT, etc.
- Python bindings: pyCUDA
- Community projects: CUSP, MAGMA, ViennaCL, etc.

CUDA

Programming in OpenMP

```
void work(double *x, double *y, double *z, int N)
{
  #pragma omp parallel
{ int thread_id = omp_get_thread_num();
  for (size_t i=thread_id; i<N; i += omp_get_num_threads())
    z[i] = x[i] + y[i];
} }
```

```
int main(int argc, char **argv)
{
  int N = atoi(argv[1]);
  double *x = malloc(N*sizeof(double));
  ...

  ...
  work(x, y, z, N); // call kernel
  ...


  free(x);
}
```

CUDA

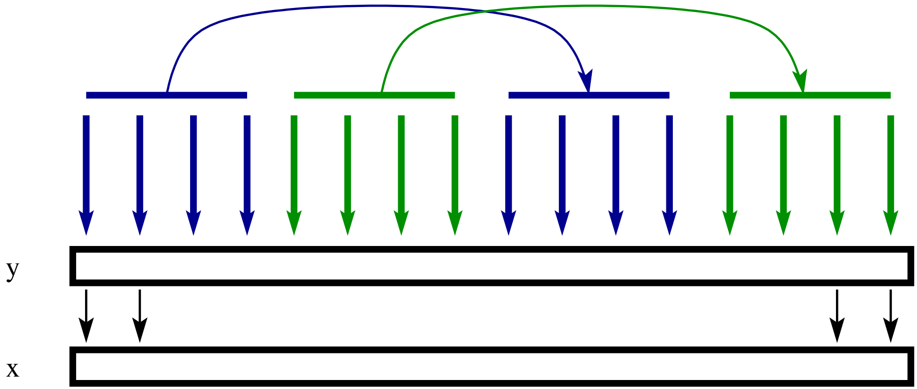
Programming in CUDA

```
__global__ void work(double *x, double *y, double *z, int N)
{

  int thread_id = blockIdx.x*blockDim.x + threadIdx.x;
  for (size_t i=thread_id; i<N; i += blockDim.x * gridDim.x)
    z[i] = x[i] + y[i];
}
```

```
int main(int argc, char **argv)
{
  int N = atoi(argv[1]);
  double *x = malloc(N*sizeof(double));
  double *gpu_x; cudaMalloc(&gpu_x, N*sizeof(double));
  cudaMemcpy(gpu_x, x, N*8, cudaMemcpyHostToDevice);
  ...
  work<<<256, 256>>>(gpu_x, gpu_y, gpu_z, N); // call kernel
  ...
  cudaMemcpy(x, gpu_x, N*8, cudaMemcpyDeviceToHost);
  ...
  free(x); cudaFree(gpu_x); // similarly for y, z, gpu_y, gpu_z
}
```

CUDA



CUDA

Thread Control (1D)

- Local ID in block: `threadIdx.x`
- Threads per block: `blockDim.x`
- ID of block: `blockIdx.x`
- No. of blocks: `gridDim.x`

Recommended Default Values

- Typical block size: 256 or 512
- Typical number of blocks: 256
- At least $10\,000$ logical threads recommended

CUDA

Memory Management

- Main memory is allocated via `malloc()`
- CUDA memory is allocated via `cudaMalloc()`
- Both work with void pointers, but reference different physical memory

Beware

- Rule 1: Do not use CPU pointers in GPU kernels
- Rule 2: Do not use GPU pointers in CPU code
- (CUDA unified memory tries to address this, but MANY pitfalls)

CUDA

Example

```
__global__ void work(int *x) { x[3] = 5; }

int main() {
  int *my_cpu_buffer = malloc(42 * sizeof(int));
  int *my_gpu_buffer; cudaMalloc(&my_gpu_buffer, 42*sizeof(int));

  my_gpu_buffer[3] = 5;  // ERROR: write from CPU to GPU memory
  work<<<1, 1>>>(my_gpu_buffer);    // OK

  my_cpu_buffer[3] = 5;  // OK
  work<<<1, 1>>>(my_cpu_buffer);    // ERROR: write to CPU memory
      in GPU kernel
}
```