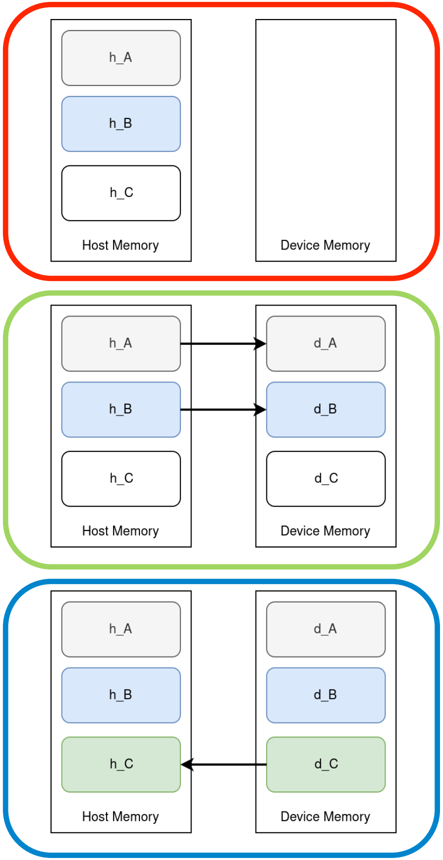
Review: Explicit Memory Management

```
… Allocate h_A, h_B, h_C …

void vecAdd(float *h_A, float *h_B, float *h_C, int n)
{
   int size = n * sizeof(float); float *d_A, *d_B, *d_C;

   cudaMalloc((void **) &d_A, size);
   cudaMalloc((void **) &d_B, size);
   cudaMalloc((void **) &d_C, size);

   cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Kernel invocation code – to be shown later

    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    cudaFree(d_A); cudaFree(d_B); cudaFree (d_C);
}

… Free h_A, h_B, h_C …
```



In Practice, Check for API Errors in Host Code

```
cudaError_t  err = cudaMalloc((void **) &d_A, size);

if (err != cudaSuccess)  {
   printf(“%s in %s at line %d\n”,   cudaGetErrorString(err), __FILE__,
   __LINE__);
   exit(EXIT_FAILURE);
}
```

NVCC Compiler

- NVIDIA provides a CUDA-C compiler
  - nvcc
- NVCC compiles device code then forwards code on to the host compiler (e.g. g++)
- Can be used to compile & link host only applications

Example 1: Hello World (main.cc)

```
#include <cstdio>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

Instructions:

1. Build and run the hello world code
2. Modify Makefile to use nvcc instead of g++
3. Rebuild and run

CUDA Example 1: Hello World

```
#include <cstdio>

__global__ void mykernel(void) {
}

int main(void) {
    mykernel<<<1,1>>>();
    printf("Hello World!\n");
    return 0;
}
```

Instructions:

1. Add kernel and kernel launch to main.cc
2. Try to build

CUDA Example 1: Build Considerations

- Build failed
  - Nvcc only parses .cu files for CUDA
- Fixes:
  - Rename main.cc to main.cu

  OR

  - nvcc –x cu
    - Treat all input files as .cu files

Instructions:

1. Rename main.cc to main.cu
2. Rebuild and Run

Compiler Flags

- Remember there are two compilers being used
  - NVCC: Device code
  - Host Compiler: C/C++ code
- NVCC supports some host compiler flags
  - If flag is unsupported, use –Xcompiler to forward to host
    - e.g. –Xcompiler –fopenmp
- Debugging Flags
  - -g: Include host debugging symbols
  - -G: Include device debugging symbols
  - -lineinfo: Include line information with symbols

CUDA-MEMCHECK

- Memory debugging tool
  - No recompilation necessary

    ```
    %> cuda-memcheck ./exe
    ```
- Can detect the following errors
  - Memory leaks
  - Memory errors (OOB, misaligned access, illegal instruction, etc)
  - Race conditions
  - Illegal Barriers
  - Uninitialized Memory
- For line numbers use the following compiler flags:
  - -Xcompiler -rdynamic -lineinfo

http://docs.nvidia.com/cuda/cuda-memcheck

Example 2: CUDA-MEMCHECK

Instructions:

1. Build & Run Example 2

   Output should be the numbers 0-9

   Do you get the correct results?
2. Run with cuda-memcheck

   ```
   %> cuda-memcheck ./a.out
   ```
3. Add nvcc flags “–Xcompiler –rdynamic –lineinfo”
4. Rebuild & Run with cuda-memcheck
5. Fix the illegal write

http://docs.nvidia.com/cuda/cuda-memcheck

CUDA-GDB

- cuda-gdb is an extension of GDB
  - Provides seamless debugging of CUDA and CPU code
- Works on Linux and Macintosh
  - For a Windows debugger use NVIDIA Nsight Eclipse Edition or Visual Studio Edition

http://docs.nvidia.com/cuda/cuda-gdb

Example 3: cuda-gdb

Instructions:

1. Run exercise 3 in cuda-gdb

   %> cuda-gdb --args ./a.out

2. Run a few cuda-gdb commands:

```
(cuda-gdb) b main                   //set break point at main
(cuda-gdb) r                        //run application
(cuda-gdb) l                        //print line context
(cuda-gdb) b foo                    //break at kernel foo
(cuda-gdb) c                        //continue
(cuda-gdb) cuda thread              //print current thread
(cuda-gdb) cuda thread 10           //switch to thread 10
(cuda-gdb) cuda block               //print current block
(cuda-gdb) cuda block 1             //switch to block 1
(cuda-gdb) d                        //delete all break points
(cuda-gdb) set cuda memcheck on     //turn on cuda memcheck
(cuda-gdb) r                                         //run from the
beginning
```

3. Fix Bug

http://docs.nvidia.com/cuda/cuda-gdb

NVPROF

Command Line Profiler

- Compute time in each kernel
- Compute memory transfer time
- Collect metrics and events
- Support complex process hierarchy's
- Collect profiles for NVIDIA Visual Profiler
- No need to recompile

Example 4: nvprof

Instructions:

1. Collect profile information for the matrix add example

   ```
   %> nvprof ./a.out
   ```

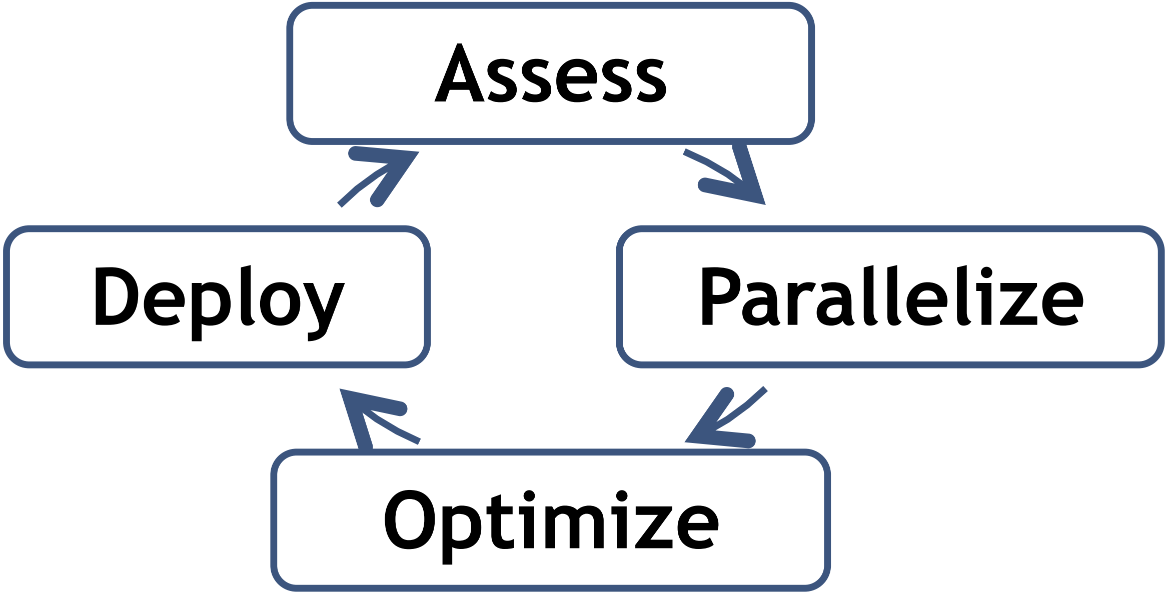
2. How much faster is add_v2 than add_v1?
3. View available metrics

   ```
   %> nvprof --query-metrics
   ```

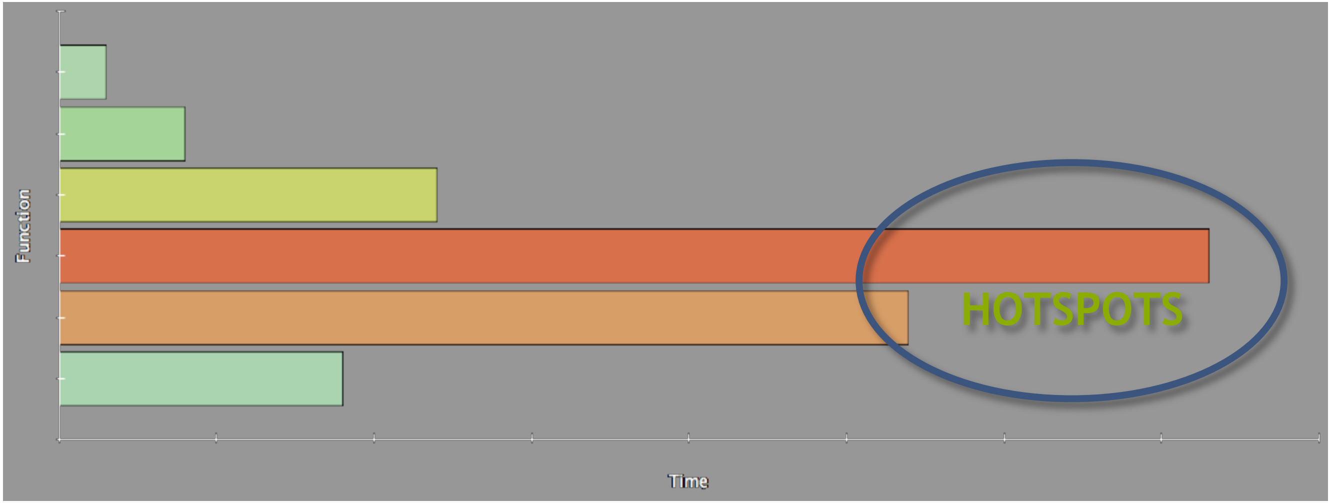
4. View global load/store efficiency

   ```
   %> nvprof --metrics gld_efficiency,gst_efficiency ./a.out
   ```

Optimization



Assess



- Profile the code, find the hotspot(s)
- Focus your attention where it will give the most benefit