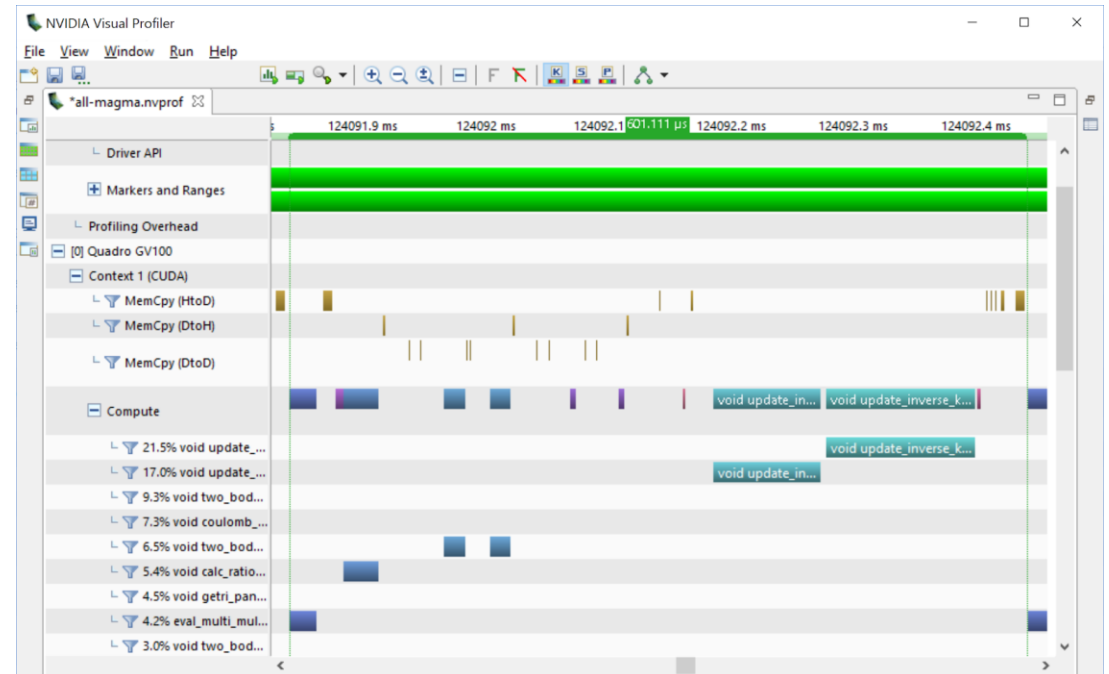


Project Introduction

- Simulation broken into 3 phases, Variadic Monte Carlo (VMC) with and without drift & Diffusion Monte Carlo (DMC)
- Each phase has similar structure with lots of very small kernels and launch latencies.
- Goal: Move a portion of these to CUDA graphs to reduce launch latencies & increase concurrency



CUDA 10.0 Graph Capture Pattern

```
cudaGraphExec_t gexec = NULL;
char graph_error[1024];
cudaError_t cuda_error;
for (int isub = 0; isub < nSubSteps; isub++)
    for (int iat = 0; iat < nat; ++iat)    {
        cudaGraph_t graph = NULL;
        cuda_error = cudaStreamBeginCapture(gpu::kernelStream, cudaStreamCaptureModeRelaxed);
        Psi.ratio(W, curr_iat, ratios_d, newG_d, newL_d);
        cuda_error = cudaStreamEndCapture(gpu::kernelStream, &graph);
        cudaGraphNode_t error_node;
        if ( (gexec == NULL) || (updateResult != cudaGraphExecUpdateSuccess) ) {
            cuda_error = cudaGetLastError(); // Clear last error
            if ( gexec != NULL ) cudaGraphExecDestroy(gexec);
            cudaGraphNode_t error_node;
            cuda_error = cudaGraphInstantiate(&gexec, graph, &error_node, graph_error, 1024);
        }
        cuda_error = cudaGraphLaunch(gexec, gpu::memoryStream);
        cuda_error = cudaEventRecord(gpu::ratioSyncDiracEvent, gpu::memoryStream);
    }
}
```

Capture *every* step.

Destroy previous executor

Instantiate *every* step

Launch *every* step.

* Extensive error checking removed for space.

CUDA 10.2 Graph Update Pattern

```
cudaGraphExec_t gexec = NULL;
char graph_error[1024];
cudaError_t cuda_error;
for (int isub = 0; isub < nSubSteps; isub++)
    for (int iat = 0; iat < nat; ++iat)    {
        cudaGraph_t graph = NULL;
        cuda_error = cudaStreamBeginCapture(gpu::kernelStream, cudaStreamCaptureModeRelaxed);
        Psi.ratio(W, curr_iat, ratios_d, newG_d, newL_d);
        cuda_error = cudaStreamEndCapture(gpu::kernelStream, &graph);
        cudaGraphNode_t error_node;
        cudaGraphExecUpdateResult updateResult;
        if ( gexec != NULL ) cuda_error = cudaGraphExecUpdate(gexec, graph, &error_node, &updateResult);
        if ( (gexec == NULL) || (updateResult != cudaGraphExecUpdateSuccess) ) {
            cuda_error = cudaGetLastError(); // Clear last error
            if ( gexec != NULL ) cudaGraphExecDestroy(gexec);
            cudaGraphNode_t error_node;
            cuda_error = cudaGraphInstantiate(&gexec, graph, &error_node, graph_error, 1024);
        }
        cuda_error = cudaGraphLaunch(gexec, gpu::memoryStream);
        cuda_error = cudaEventRecord(gpu::ratioSyncDiracEvent, gpu::memoryStream);
    }
}
```

Capture *every* step.

Attempt to update & instantiate as necessary

Instantiate only when necessary

Launch *every* step.

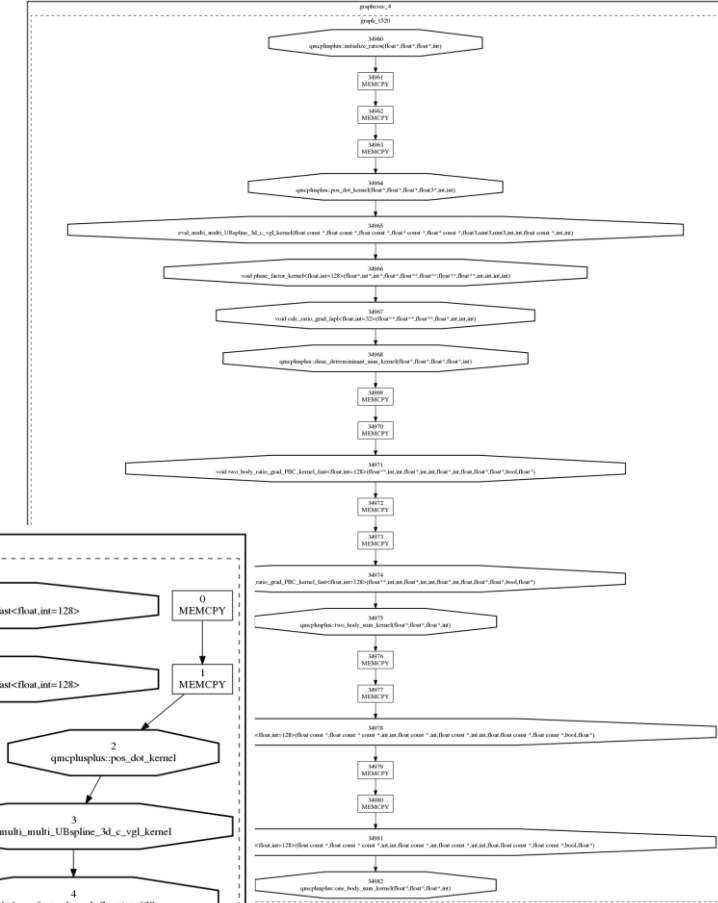
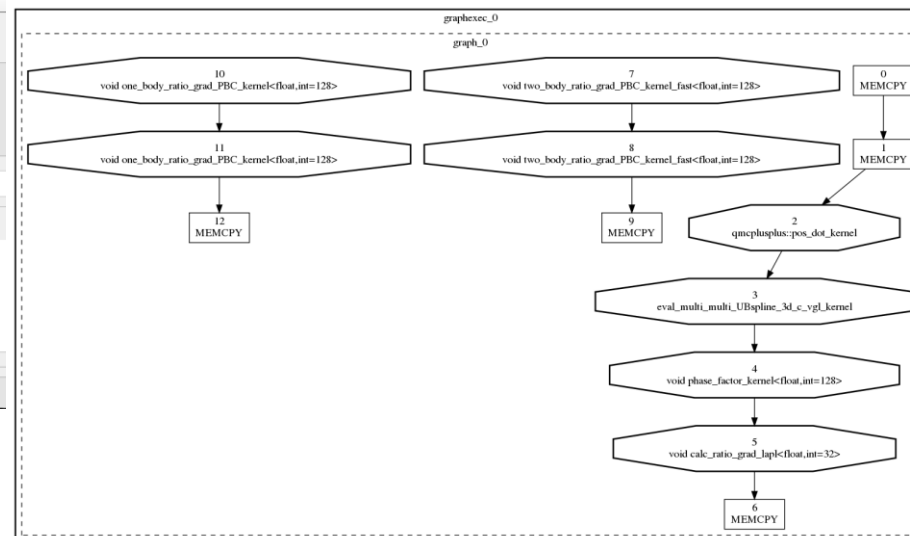
* Extensive error checking removed for space.

Nsight Compute Graph Visualization

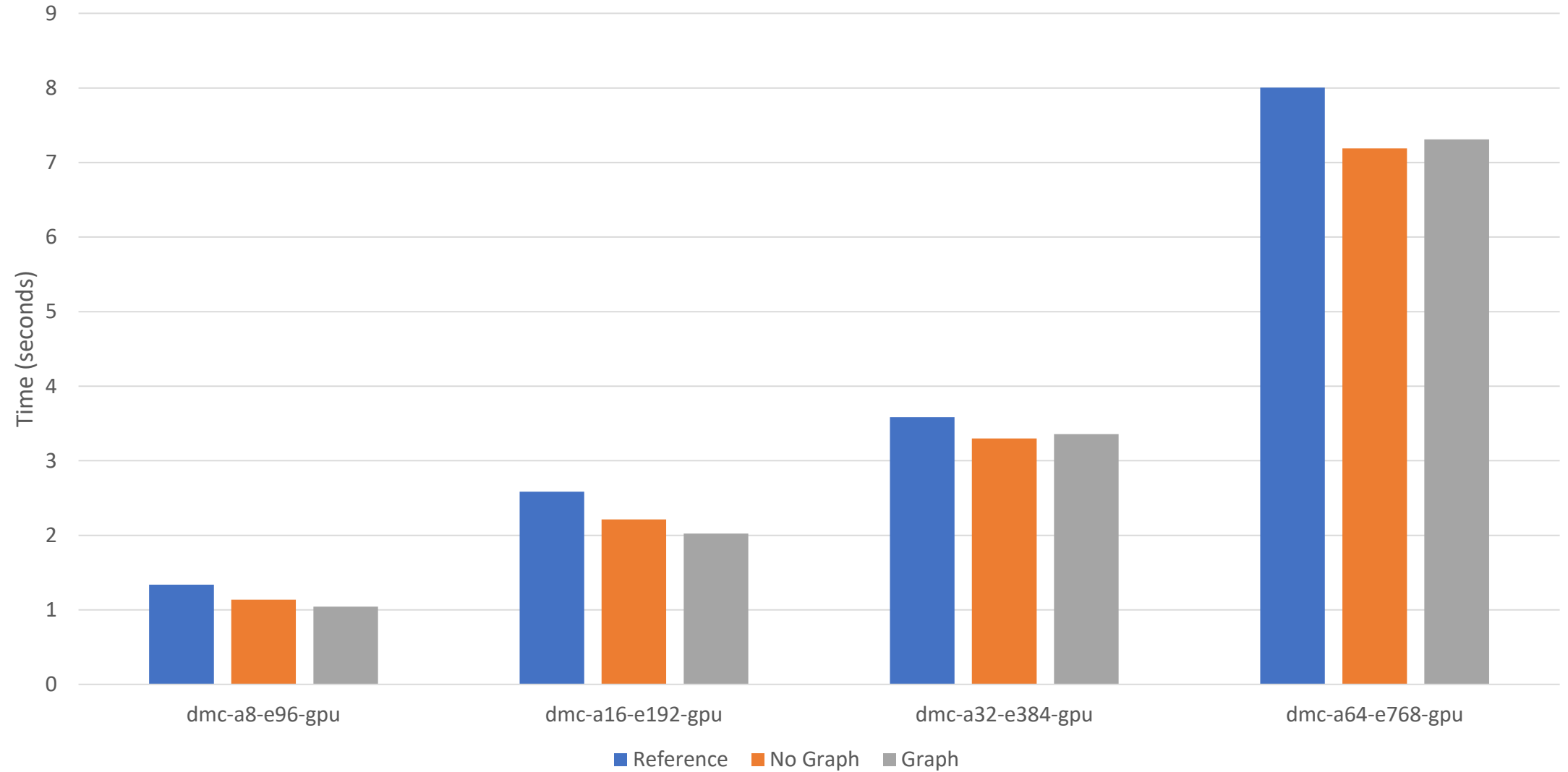
The screenshot displays the NVIDIA Nsight Compute application window. The top menu bar includes File, Connection, Debug, Profile, Tools, Window, and Help. Below the menu is a toolbar with icons for Connect, Disconnect, Terminate, Profile Kernel, and various execution controls. The main interface is divided into several panels:

- API Stream:** Shows the current API call being executed. The "Next Trigger" is set to "cudaGraphLaunch". A blue callout box labeled "1. Trigger on cudaGraphLaunch" points to this field. The "Run To Next API Call" button is highlighted with a blue callout box labeled "2. Run To Next API Call".
- API Statistics:** Displays a table of API statistics. The table has columns for Name, Number of Calls, Total Duration, Average Duration, Minimum Duration, and Maximum Duration. The first row shows "cudaThreadExchangeStreamCaptureMode" with 1132 calls and a total duration of 28.933ms. A blue callout box labeled "3. Export to SVG or GraphViz (dot)" points to the "Export to SVG" button in the Resources panel.
- Resources:** Shows a list of resources, including CUgraphs. The first row shows a CUgraph with ID 0x5628dfd53ce8, API Call # 91312, and 23 nodes. A blue callout box labeled "1. Trigger on cudaGraphLaunch" points to the "Next Trigger" field in the API Stream panel.

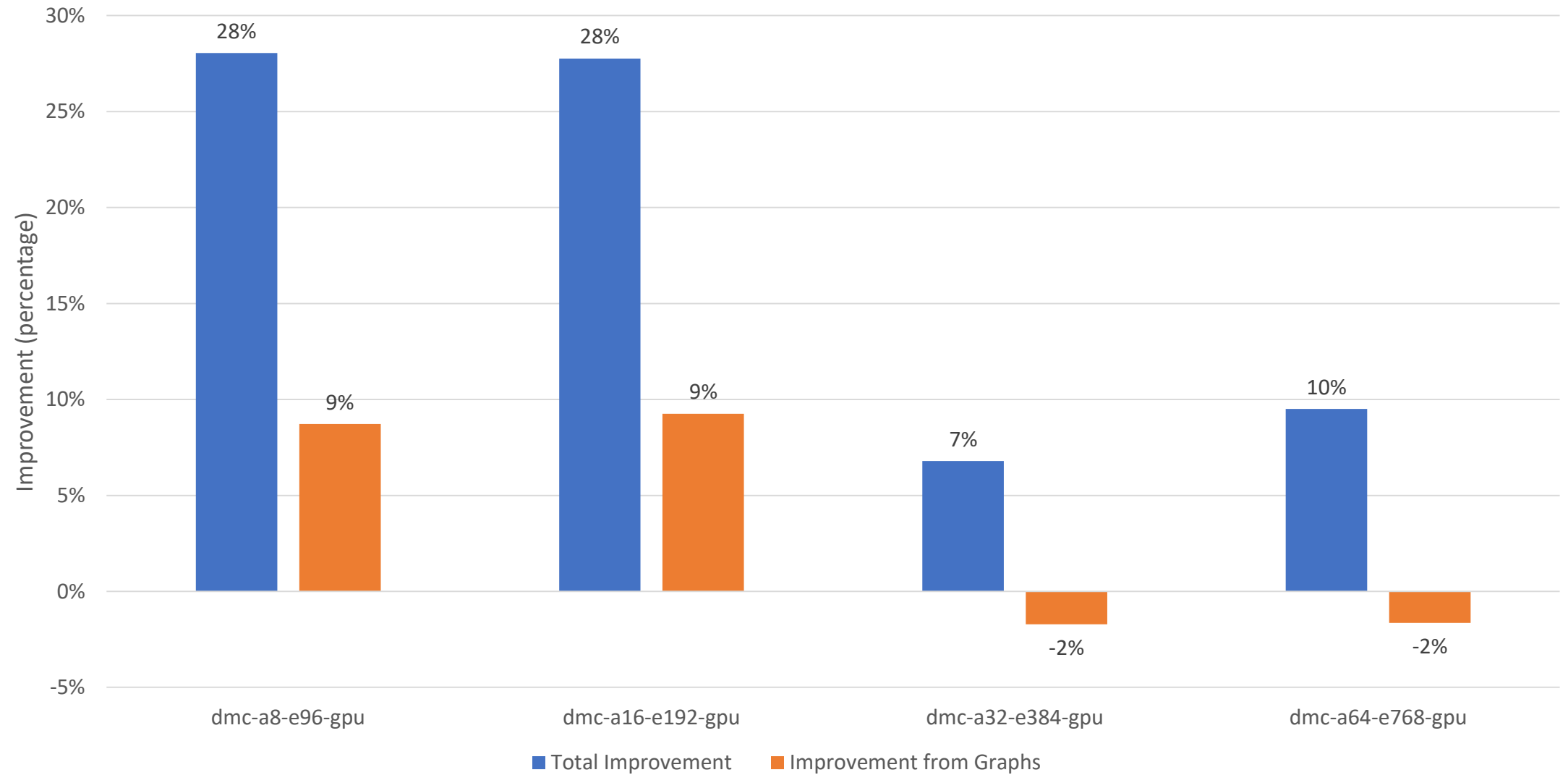
On the right side of the interface, there is a graph visualization showing the execution flow of the API calls. The graph consists of nodes representing API calls and edges representing the sequence of execution. The nodes are labeled with their IDs and names, such as "void one_body_ratio_grad_PBC_kernel<float,int=128>" and "void two_body_ratio_grad_PBC_kernel<float,int=128>". The graph is titled "graphsec_0" and "graph_0".



NiO Benchmark - VMC w/o Drift Phase



NiO Benchmark - VMC w/o Drift Phase



Lessons I've Learned (So Far)

- Graph Update makes CUDA Graphs *significantly* simpler
 - Didn't necessitate moving as much from the CPU for successful capture.
 - Only instantiates when a structural change has occurred
 - NOTE: I couldn't always tell what changed via Nsight graph
- Application had WAY more reliance on default stream than I expected
 - Avoid the default stream, expose as much concurrency as possible.
- Error check OFTEN
 - Errors during capture crop up much later, cudaGetLastError is your friend
 - Not returning cudaSuccess shouldn't always cause an abort