Variante mit abgebildetem Speicher

gemm-mapped-and-streamed.cu

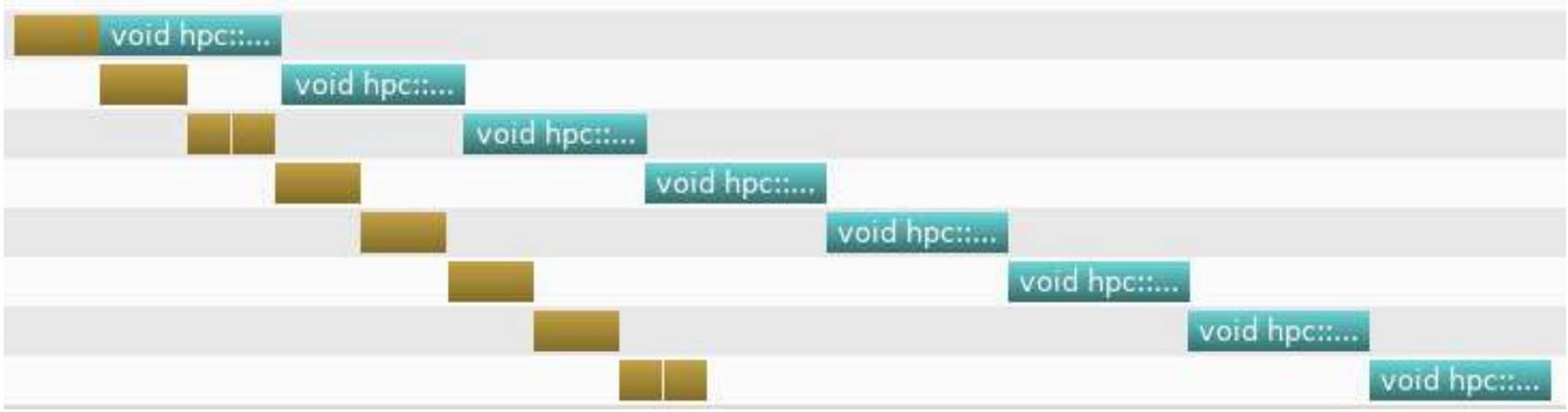
```
std::unique_ptr<GeMatrix<T>> A_host[COUNT];
std::unique_ptr<GeMatrix<T>> B_host[COUNT];
std::unique_ptr<DeviceGeMatrix<T>> A_dev[COUNT];
std::unique_ptr<DeviceGeMatrix<T>> B_dev[COUNT];
std::unique_ptr<MappedGeMatrix<T>> C[COUNT];

for (std::size_t index = 0; index < COUNT; ++index) {
   A_host[index] = std::make_unique<GeMatrix<T>>(M, K, Order::RowMajor);
   B_host[index] = std::make_unique<GeMatrix<T>>(K, N, Order::RowMajor);
   C[index] = std::make_unique<MappedGeMatrix<T>>(M, N, Order::RowMajor);
   init_matrix(*A_host[index]);
   init_matrix(*B_host[index]);
   init_matrix(*C[index]);
   A_dev[index] = std::make_unique<DeviceGeMatrix<T>>(M, K, Order::RowMajor);
   B_dev[index] = std::make_unique<DeviceGeMatrix<T>>(K, N, Order::RowMajor);
}

auto start = std::chrono::high_resolution_clock::now();
Stream stream[COUNT];
for (std::size_t index = 0; index < COUNT; ++index) {
   copy(*A_host[index], *A_dev[index], stream[index]);
   copy(*B_host[index], *B_dev[index], stream[index]);
   cuda_gemm(alpha, *A_dev[index], *B_dev[index], beta, *C[index],
      stream[index]);
}
CHECK_CUDA(cudaDeviceSynchronize); // wait for the kernel to finish
auto finish = std::chrono::high_resolution_clock::now();
```

Gemessene Zeiten


```
livingstone$ nvprof --print-gpu-summary gemm-streamed 2>&1 | cut -b1-104
==24807== NVPROF is profiling process 24807, command: gemm-streamed
8 gemm operations on gpu took 0.105245 s for (1024 x 1024) (1024 x 1024)
time per gemm operation: 0.0131556 s
==24807== Profiling application: gemm-streamed
==24807== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   51.81%  94.515ms         8  11.814ms  11.777ms  11.832ms  void hpc::cuda::gemm_kernel
                   36.77%  67.082ms        24  2.7951ms  2.7799ms  3.0637ms  [CUDA memcpy HtoD]
                   11.42%  20.829ms         8  2.6036ms  2.5577ms  2.9210ms  [CUDA memcpy DtoH]
livingstone$ nvprof --print-gpu-summary gemm-mapped-and-streamed 2>&1 | cut -b1-104
==24821== NVPROF is profiling process 24821, command: gemm-mapped-and-streamed
8 gemm operations on gpu took 0.0940759 s for (1024 x 1024) (1024 x 1024)
time per gemm operation: 0.0117595 s
==24821== Profiling application: gemm-mapped-and-streamed
==24821== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   66.23%  88.828ms         8  11.104ms  11.064ms  11.130ms  void hpc::cuda::gemm_kernel
                   33.77%  45.283ms        16  2.8302ms  2.7809ms  2.9374ms  [CUDA memcpy HtoD]
livingstone$
```



- Die expliziten asynchronen Kopieraktionen *device to host* sind jetzt weggefallen.
- Entsprechend wird es jetzt etwas schneller fertig, da wir nicht mehr auf eine letzte Kopieraktion warten müssen.
- Wichtig ist, dass nur C abgebildet wird. Wenn A und B ebenfalls abgebildet wären, dann würde das zu katastrophalen Zeiten führen.

Die CUDA-Schnittstelle bietet auch die Möglichkeit, auf konventionelle Weise belegten Speicher (etwa mit **new** oder *malloc*) nachträglich gegen Auslagerung zu schützen:

cudaError_t cudaHostRegister(**void**∗ *ptr*, *size_t size*, **unsigned int** *flags*)
schützt die Speicherfläche, auf die *ptr* verweist, vor einer Auslagerung. Zwei Optionen werden unterstützt:

cudaHostRegisterPortable die Speicherfläche wird von allen GPUs als nicht auslagerbar erkannt.

cudaHostRegisterMapped die Speicherfläche wird in den Adressraum der GPU abgebildet. (Achtung: Selbst bei UVM kann nicht auf *cudaHostGetDevicePointer* verzichtet werden.)

cudaError_t cudaHostUnregister(**void**∗ *ptr*)
beendet den Schutz vor Auslagerung.

Die CUDA-Schnittstelle unterstützt Ereignisse, die der Synchronisierung und der Zeitmessung dienen:

cudaError_t cudaEventCreate(cudaEvent_t event)*
: legt ein Ereignis-Objekt an mit der Option *cudaEventDefault*, d.h. Zeitmessungen sind möglich, eine Synchronisierung erfolgt jedoch im *busy-wait*-Verfahren.

cudaError_t cudaEventRecord(cudaEvent_t event, cudaStream_t stream)
: fügt in die zu *stream* gehörende Ausführungssequenz die Anweisung hinzu, das Eintreten des Ereignisses zu signalisieren.

cudaError_t cudaEventDestroy(cudaEvent_t event)
: gibt die mit dem Ereignis-Objekt verbundenen Ressourcen wieder frei.

CUDA-Event-Operationen zur Synchronisierung und Zeitmessung:

cudaError_t cudaEventSynchronize(cudaEvent_t event)

der aufrufende Thread wartet, bis das Ereignis eingetreten ist. Wenn jedoch *cudaEventRecord* vorher noch nicht aufgerufen worden ist, kehrt dieser Aufruf sofort zurück. Wenn die Option *cudaEventBlockingSync* nicht gesetzt wurde, wird im *busy-wait*-Verfahren gewartet.

*cudaError_t cudaEventElapsedTime(***float**** ms, cudaEvent_t start, cudaEvent_t end)*

liefert die Zeit in Millisekunden, die zwischen den Ereignissen *start* und *end* vergangen ist. Die Auflösung der Zeit beträgt etwa eine halbe Mikrosekunde. Diese Zeitmessung ist genauer als konventionelle Methoden, weil hierfür die Uhr auf der GPU verwendet wird.

Zeitmessung mit CUDA-Events


```
cudaEvent_t start_event; CHECK_CUDA(cudaEventCreate, &start_event);
cudaEvent_t end_event; CHECK_CUDA(cudaEventCreate, &end_event);
CHECK_CUDA(cudaEventRecord, start_event);

// kernel invocations, data transfers etc.

CHECK_CUDA(cudaEventRecord, end_event);
CHECK_CUDA(cudaDeviceSynchronize); // wait for everything to finish

float timeInMillisecs;
CHECK_CUDA(cudaEventElapsedTime, &timeInMillisecs,
   start_event, end_event);
std::cerr << "GPU time in ms: " << timeInMillisecs << std::endl;
CHECK_CUDA(cudaEventDestroy, start_event);
CHECK_CUDA(cudaEventDestroy, end_event);
```

- Zu beachten ist hier, dass *cudaEventRecord* asynchron abgewickelt wird und das Ereignis erst signalisiert, wenn alle laufenden CUDA-Operationen abgeschlossen sind. Die CPU muss sich also danach immer noch mit *cudaDeviceSynchronize* synchronisieren.
- Zeitmessungen sollten immer auf Ereignissen beruhen, die mit dem NULL-Stream verbunden sind. Das Messen der Realzeit einzelner CUDA-Streams ist nicht sinnvoll, da sich jederzeit andere Operationen dazwischenschieben können.